

# NECMETTİN ERBAKAN ÜNİVERSİTESİ

Mühendislik Fakültesi



STAJ DEFTERİ

ÖĞRENCİNİN	
BÖLÜMÜ	:
ADI, SOYADI	:
ÖĞRENCİ NUMARASI :	

# **NECMETTİN ERBAKAN ÜNİVERSİTESİ**

## **MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ**

### **STAJ ESASLARI**

1. Her öğrenci ders dağılım çizelgesinde yer alan tüm stajları başarı ile tamamlamak zorundadır.
2. Öğrenci staj süresince Necmettin Erbakan Üniversitesi Disiplin yönetmeliği ve işyerinin özel kurallarına uymak zorundadır.
3. Öğrencinin kendi bulduğu staj yerinde en az bir mühendis/mimar çalışmalıdır. Görevli mühendisin/mimarın branşı ile öğrencinin bölümü uyumlu olmalıdır. Staj yerinin uygunluğuna Bölüm Staj Komisyonu karar verdikten sonra öğrenci staja başlayabilir. İşyerinin değerlendirilmesinde öğrenci ve işyerinden alınan bilgilere göre karar verilir. Gerçeğe aykırı bilgi ve beyan tespit edildiğinde staj durdurulur.
4. Yaz döneminde staj yapan öğrenci staj defterini güz yarıyılıının ilk iki haftası içerisinde, öğretim yılı içinde staj yapan öğrenci ise stajın bitim tarihinden itibaren iki hafta içinde Bölüm Staj Komisyonu Başkanlığına teslim etmek zorundadır.
5. Stajın değerlendirilmesi Bölüm Staj Komisyonu tarafından görüşmeye tabi tutularak yapılır. Görüşmede staj defterleri ve ilgili raporlar incelenir ve stajın başarılı veya başarısız olduğuna karar verilir ve durum Dekanlığa bildirilir.
6. Her staj sürekli olup, ait olduğu dönemde ve bir defada yapılır. Mazereti sebebi ile staj süresince devamsızlık yapan öğrenci devam edemediği gün kadar stajı tamamlamak zorundadır. Toplam staj süresinin %20 sine devam etmeyen öğrencinin stajı başarısız kabul edilerek bir sonraki yılda staj tekrar yapılır. Staj süresi eksik olduğu takdirde staj başarısız kabul edilir.
7. Staj yerlerinde yapılacak işlemlerle ilgili ayrıntılı bilgileri içeren yönergeler, öğrencinin staj yapacağı kuruma öğrenciler tarafından iletmek zorundadır.
8. Staj esnasında yapılan tüm işlemlerin teknik detayı gerekli resim, hesaplar ve açıklamalar temiz bir şekilde staj defteri için verilen örnek sayfadan gerektiği kadar çoğaltılarak kaydedilir. Stajın bitiminde tüm sayfalar ve belgeler tek bir dosya halinde teslim edilir.
9. Staj defteri içine konulacak kağıda basılı belgeler A4 kağıdı büyüklüğünde olmalı veya A4 boyutlarında katlanmış olmalıdır.
10. Öğrenci çalışma konusunu tanıtmak üzere CD, DVD, fotoğraf, fotokopi, ozalit, rapor, proje vs. gibi dokümanları staj defterine ekleyebilir. Elektronik ortamda bulunan belgelerin stajın değerlendirilmesi esnasında açılmaması durumunda belge yok kabul edilecektir.
11. Staj defterinde belirtilen kısımlar eksiksiz doldurulup, ilgili sorumlulara imzalatılır. Defter iş yeri yetkilisi tarafından onaylanır.
12. Staj sicil fişi, iş yeri yetkilileri tarafından doldurulup tasdik edilerek, ilgili Bölüm Başkanlığına posta ile kapalı zarf ile gönderilir.
13. Bölüm staj komisyonu gerek gördüğü takdirde öğrencileri staj yerinde denetler.
14. Kabul veya reddedilen stajlar Bölüm Başkanlığınca ilan edilir.

**Adı Soyadı : HASAN KARAYAKA**  
**Numara : 21100101029**  
**Staj Yeri : OTOBOT**

### STAJ DEVAM ÇİZELGESİ

Sıra No	Tarih	Çalışılan Kısım	Yetkili İmza
1.	08/07/2024	AKİBA AR-GE	
2.	09/07/2024	AKİBA AR-GE	
3.	10/07/2024	AKİBA AR-GE	
4.	11/07/2024	AKİBA AR-GE	
5.	12/07/2024	AKİBA AR-GE	
6.	16/07/2024	AKİBA AR-GE	
7.	17/07/2024	AKİBA AR-GE	
8.	18/07/2024	AKİBA AR-GE	
9.	19/07/2024	AKİBA AR-GE	
10.	22/07/2024	AKİBA AR-GE	
11.	23/07/2024	AKİBA AR-GE	
12.	24/07/2024	AKİBA AR-GE	
13.	25/07/2024	AKİBA AR-GE	
14.	26/07/2024	AKİBA AR-GE	
15.	29/07/2024	AKİBA AR-GE	
16.	30/07/2024	AKİBA AR-GE	
17.	31/07/2024	AKİBA AR-GE	
18.	1/08/2024	AKİBA AR-GE	
19.	2/08/2024	AKİBA AR-GE	
20.	5/08/2024	AKİBA AR-GE	
21.	6/08/2024	AKİBA AR-GE	
22.	7/08/2024	AKİBA AR-GE	
23.	8/08/2024	AKİBA AR-GE	
24.	9/08/2024	AKİBA AR-GE	
25.	12/08/2024	AKİBA AR-GE	
26.	13/08/2024	AKİBA AR-GE	
27.	14/08/2024	AKİBA AR-GE	
28.	15/08/2024	AKİBA AR-GE	
29.	16/08/2024	AKİBA AR-GE	
30.	19/08/2024	AKİBA AR-GE	

**CAN BUS nedir?**

- Uygulama alanı yüksek hızlı ağlardan düşük maliyetli çoklu kablolamalı sistemlere kadar genişler.
- CANBUS otomobil elektroniği, akıllı motor kontrolü, robot kontrolü, akıllı sensörler, asansörler, makine kontrol birimleri, kaymayı engelleyici sistemler, trafik sinyalizasyon sistemleri, akıllı binalar ve laboratuvar otomasyonu gibi uygulama alanlarında maksimum 1Mbit/sn'lik bir haber veri iletişimi sağlar.
- İletişim hızı 40m'de 1Mbit/sn iken 1km uzaklıklarda 40Kbit/sn'ye düşmektedir.
- CAN diğer protokollerden farklı olarak adres temelli değil mesaj temelli çalışmaktadır.
- Her mesaja özgü bir ID numarası vardır. Mesajlar çerçeveler ile iletilirler.

**CAN BUS**

- Nesne Katmanı
- İletim Katmanı
- Fiziksel Katman

**Nesne Katmanının Görevi**

Hangi mesajın transfer edileceğini tespit etmek

İletim katmanında hangi mesajın alınacağına karar vermek

Donanımla ilgili uygulamaya arayüz sağlamaktır.

**İletim Katmanının Görevi**

İletim Katmanının Başlıca görevi transfer protokolüdür.

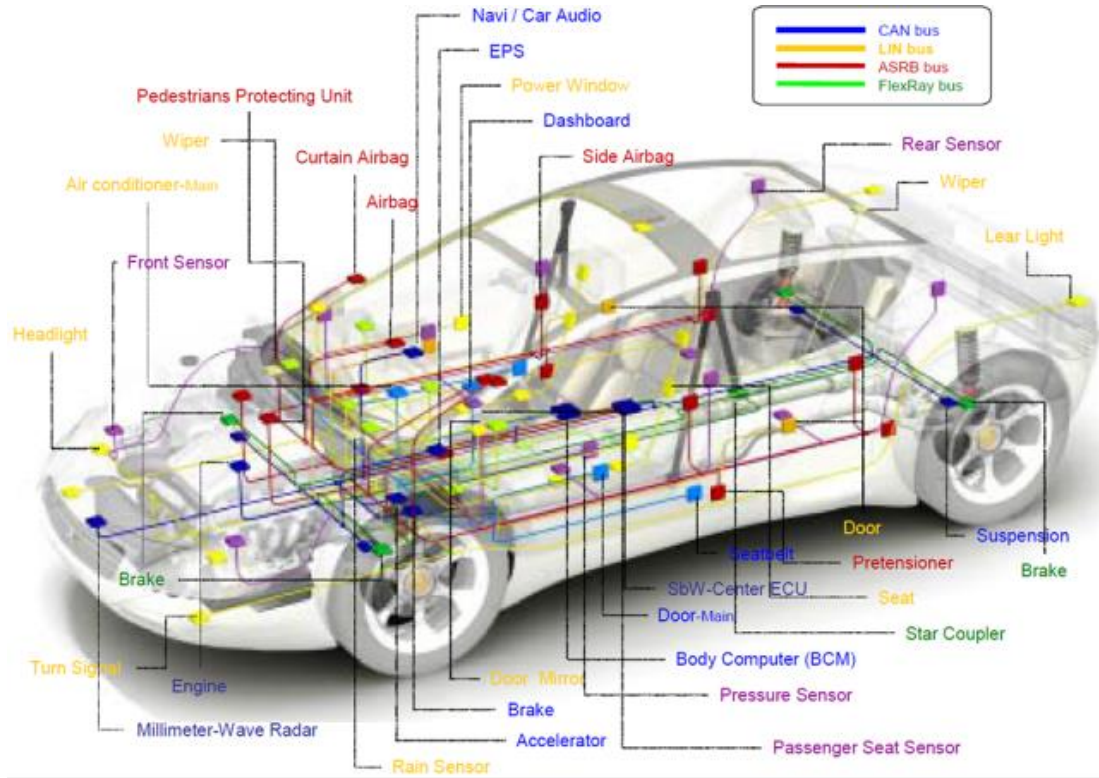
Çerçeve(frame) kontrolü, mesaj önceliği belirleme, hata kontrolü, hata sinyali ve hata kapatma

**Fiziksel Katmanın Görevi**

Üniteler arasındaki veri haberleşmesi sırasındaki tüm elektriksel kısımdır.

**CAN BUS SİSTEMİNİN GENEL ÖZELLİKLERİ**

- Mesaj Önceliği
- Kayıp Zaman Güvenliği
- Yapılandırma Esnekliği
- Senkronizasyonlu çoklu kabul: Aynı veri birçok ünite tarafından alınabilir.
- Sistemdeki veri yoğunluğunu kaldırabilme
- Çok efendili ( Multi master ) çalışma
- Hata tespiti ve hataya ilişkin sinyalleri üretme
- Mesaj yollanmasında hata oluşması halinde mesajın iletim hattının (BUS) boş olduğu bir anda mesajın otomatik olarak tekrar yollanması
- Ünitelerde oluşan geçici ve kalıcı hataları ayırt edebilme ve özerk olarak kalıcı hatalı üniteleri kapatabilme

**Şekil 1:** Otomobildeki CAN BUS hattı.

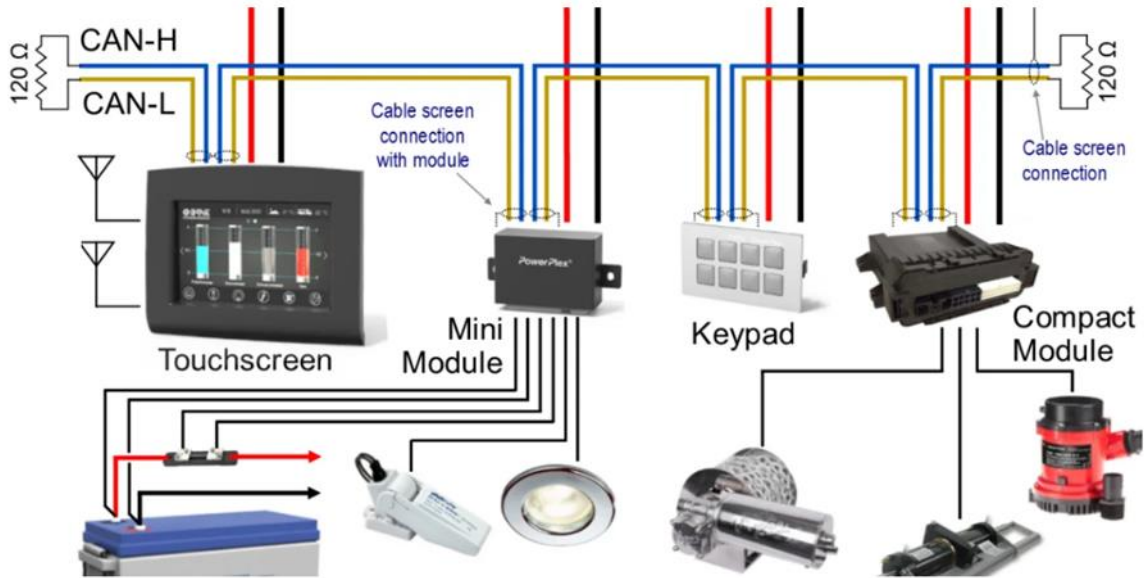
## CAN-BUS ÇALIŞMA MANTIĞI

CANBUS sisteminde tüm üniteler eşit öncelikli olarak iletim hattına veri yollama hakkına sahiptir. Buna multimaster çalışma denir.

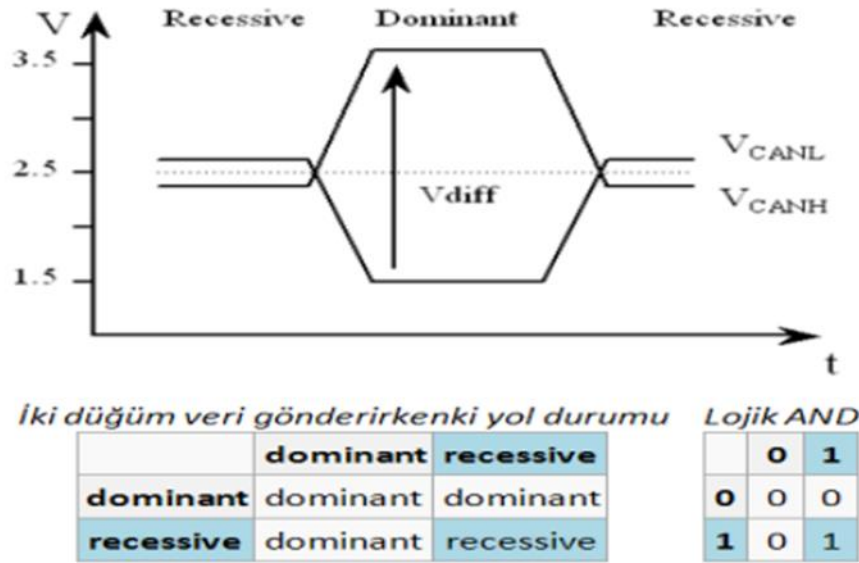
Peki CANBUS'ta hatasız veri iletimi nasıl sağlanır?

Tüm üniteler aynı anda veri yollamaya çalışırsa çatışmalar olacağı açıktır. Bunun çözümü, tüm ünitelerin iletim hattını sürekli dinlemesi ve hattın boş olduğu anı yakalayıp verisini yollamasıdır. Her ne kadar tüm ünitelerin eşit mesaj yollama önceliği olsa da aslında durum farklıdır. Bunun sebebi, CAN'in mesaj öncelikli bir sistem olmasından kaynaklanır. İnternet sisteminde PC'lere numara verilirken, CANBUS'ta ünitelere değil, mesajlara numara verilir.

Örneğin bir araç hızla giderken karşıdan gelen araca çarpma durumu oluştuğunda, şoför frene basar ve burada CAN'in önemi devreye girer. Örneğin araç önden çarpmış ve sensörler bunu fark ettiğinde, hava yastıklarının açılması, yakıtın kesilmesi gibi bir dizi önlem alınmalıdır. Bu işlemlerden sorumlu tüm ünitelere 1 numaralı mesaj (kaza oldu, güvenlik sistemleri devreye girsin) iletilmelidir. Bu mesajı sensörlü ünite, diğer ünitelere iletir. Ancak o sırada başka bir ünite, örneğin motor ısısının kaç derece olduğunu kokpitteki LCD'ye yollamaya çalışıyorsa, öncelikli mesajların daha önce iletim ortamına iletilmesi sağlanır. CAN, tek kablo üzerinden eşit erişimli mesaj yollanmasına izin verir, ancak öncelikli mesajların öncelikli olarak iletilmesine de özen gösterir.



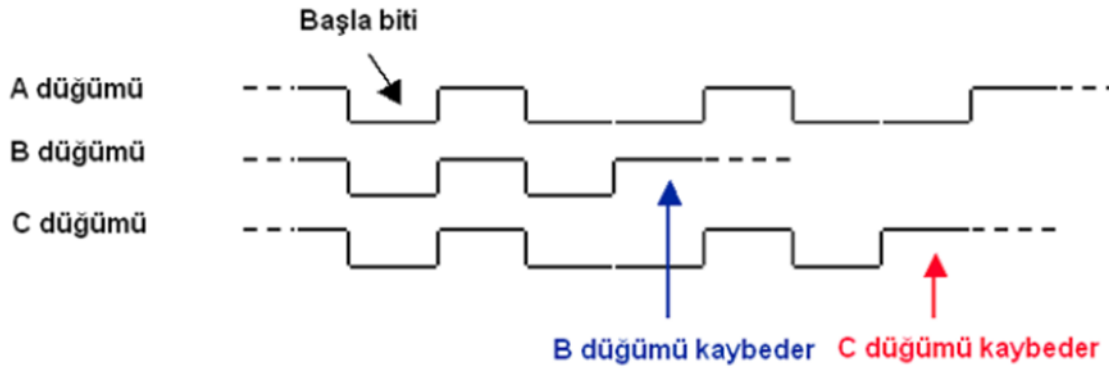
Şekil 2: Örnek Bağlantı Haritası



Şekil 3: CAN-H CAN-L Grafiği

- Yukardaki şekilde görüldüğü gibi hattın lojik seviyesi 2 farklı değer alabilmektedir ve bu değerlerin seviyesi gözükmemektedir
- Lojik1 recessive (çekingen) , lojik0 dominant (baskın) olarak adlandırılır.
- Bunun sebebi hatta farklı düğümlerden aynı anda 0 ve 1 yazılması durumunda 0'ın 1'e karşı baskın gelmesidir.
- Bu durumun doğruluk grafiği yukarda verilmiştir.
- Eğer can hattından veri gelmiyorsa multimetre yardımı ile canh ve canl kanalları arasındaki volt farkı bize mesajın gönderilip gönderilmediği hakkında bilgi verecektir.

- Lojik 0 ın lojik 1e baskın gelmesi sonucu küçük mesaj ID sine sahip mesajlar öncelik kazanırlar.
  - Bir düğüm tarafından mesaj gönderilmesi kararlaştırıldığında mesaj yol boşalana kadar bekletilir.
  - Her düğüm yolu devamlı izlemektedir.
  - Yol boşaldıktan sonra düğüm yola başla işaretini vererek mesajı yollamaya başlar.
  - Mesaj her düğüme ulaşmaktadır ve ilişkisi olan düğümler mesajı okuyup işlemektedirler.
  - Eğer yol boşaldığında birden fazla düğüm yola mesaj yazmaya başlarsa düşük ID li mesajı yazan düğüm yolu ele geçirir ve diğer düğümler aradan çekilerek tekrar göndermek üzere yolun boş almasını beklerler.
  - Bu mekanizma şu şekilde çalışır.
  - Yazılan her bitin aynı anda okunduğundan bahsetmiştik.
  - Bir düğüm veri yoluna mesaj yazarken 1 yazdığında 0 okuyorsa eğer, başka bir düğümünde yola mesaj yazdığını anlar ve onun önceliği yüksek olduğundan veri yolunu ona bırakır.
  - Yol boşaldığında tekrar göndermeye çalışır. >
- Örneğin yola aynı anda veri yazmaya çalışan A, B ve C adında 3 düğümümüz olsun. >  
A düğümü yola 36 (100100), B düğümü 47(101111) ve C düğümü 37(100101) yazsın. >  
Aşağıdaki şekilde bu durum gösterilmiştir.

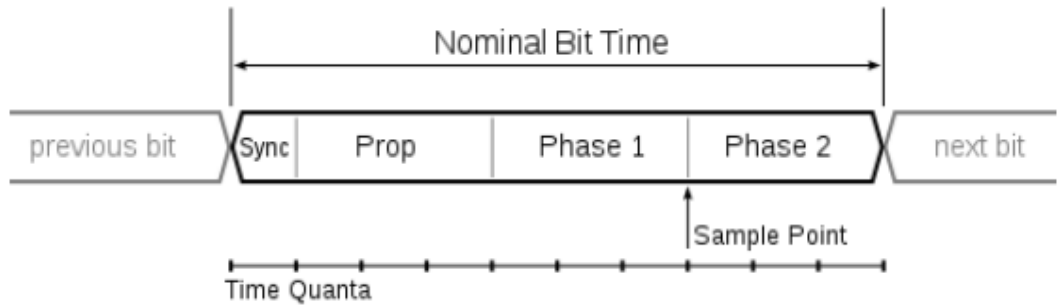


Şekil 4: Birden fazla can mesajı grafiği



**BIT SÜRESİ HESAPLAMA**

- Çoğu diğer seri protokolün aksine, CAN protokolünde bit hızı direk olarak baud rate önbölücüsünü kurarak ayarlanmaz.
- CAN donanımlarında baud rate önbölücüsü vardır fakat kuantta denilen küçük bir zaman dilimini üretmek için kullanılır.
- Bir bitlik süre 3 kısma bölünmüştür.
- Birinci kısım senkronizasyon kısmıdır ve sabit olarak bir kuantta uzunluğundadır.
- Takip eden kısımlar ise Tseg1 ve Tseg2 olarak isimlendirilir ve kullanıcı tarafından uzunlukları kuantta cinsinden ayarlanabilir.
- Bir bitlik periyot minimum 8 maksimum 25 kuantta uzunluğunda olmalıdır.
- Gönderilen bitin alıcıda alındığı nokta örnekleme noktası diye isimlendirilir ve Tseg1 sonundadır

**Şekil 5: Nominal Bit Time**

- Tseg1 ve Tseg2 oranı ayarlanarak örnekleme noktası bir bitlik zaman içerisinde kaydırılabilir.
- Bunu yapmamızdaki amaç iletim hattının uzunluğuna göre sistemin kararlı çalışabilmesini sağlamak-tır.Uzun iletim hatları kullanıyorsak örnekleme noktası geri çekilmelidir.
- Osilatörümüz hassas değil ve kesinliği düşük ise örnekleme noktası ileri kaydırılır.
- Ek olarak alıcılar bit zamanlamalarını ayarlayarak vericiye kilitlenebilirler.
- Bu vericinin bit hızındaki ufak sapmaları telafi eder.
- Her bit, kullanıcı tarafından ayarlanabilen, synchronous jump width denilen, 1-4 kuantta süresi arasında değer alan bir değişken tarafından ayarlanır.
- Bit hızı aşağıdaki bağıntı ile hesaplanır(BRP=Baud Rate Presaler).

- Bit Rate =  $PCLK / ( BRP * ( 1 + Tseg1 + Tseg2 ) )$
- Bu bağıntı birkaç bilinmeyene sahiptir.
- Bit hızını 125Khz, PCLK yı 60Mhz ve örnekleme noktasını %70 olarak kullandığımızı varsayalım.
- Bir bitlik periyot toplam kuanta sayısı ile hesaplanır ve bu değer  $(1+Tseg1+Tseg2)$  dir.
- Bu değere KUANTA diyelim ve yukarıdaki bağıntıyı tekrar düzenleyelim.
- $BRP = PCLK / ( Bit Rate * KUANTA )$
- Bilinen değerlerimizi denklemde yerine koyalım.
- $BRP = 60M / ( 125K * KUANTA )$
- Bir bitlik periyodun 8 ile 25 kuanta arasında olduğunu biliyoruz.
- Bu bilgiyi kullanarak BRP tam sayı olacak şekilde KUANTA yerine 8 ile 25 arasında uygun bir sayı seçelim.
- KUANTA =16 , BRP=30 olacak şekilde denklemi çözeriz.
- Şimdi Tseg1 ile Tseg2 arasındaki oranı ayarlayalım.
- $16 = (1+Tseg1+Tseg2)$  olduğuna göre hedeflenen örnekleme noktasının periyodun %70 ine denk gelmesi için
- Örnekleme Noktası =  $(KUANTA * 70) / 100$
- Dolayısıyla  $16 * 0.7 = 11.2$  olur.
- Buradan Tseg1 = 10 ve Tseg2 = 5 olarak bulunur.Bu durumda örnekleme noktası %68.8 e denk gelir.
- Synchronous jump width değeride aşağıdaki şekilde hesaplanır.
- $Tseg2 \geq 5 TKUANTA$  ise SJW =4 tür.
- $Tseg2 < 5 TKUANTA$  ise SJW =  $( Tseg2 - 1 ) TKUANTA$  dır.
- Bizim örneğimizde SJW=4 tür.

**CAN BUS MESAJ ALIMI**

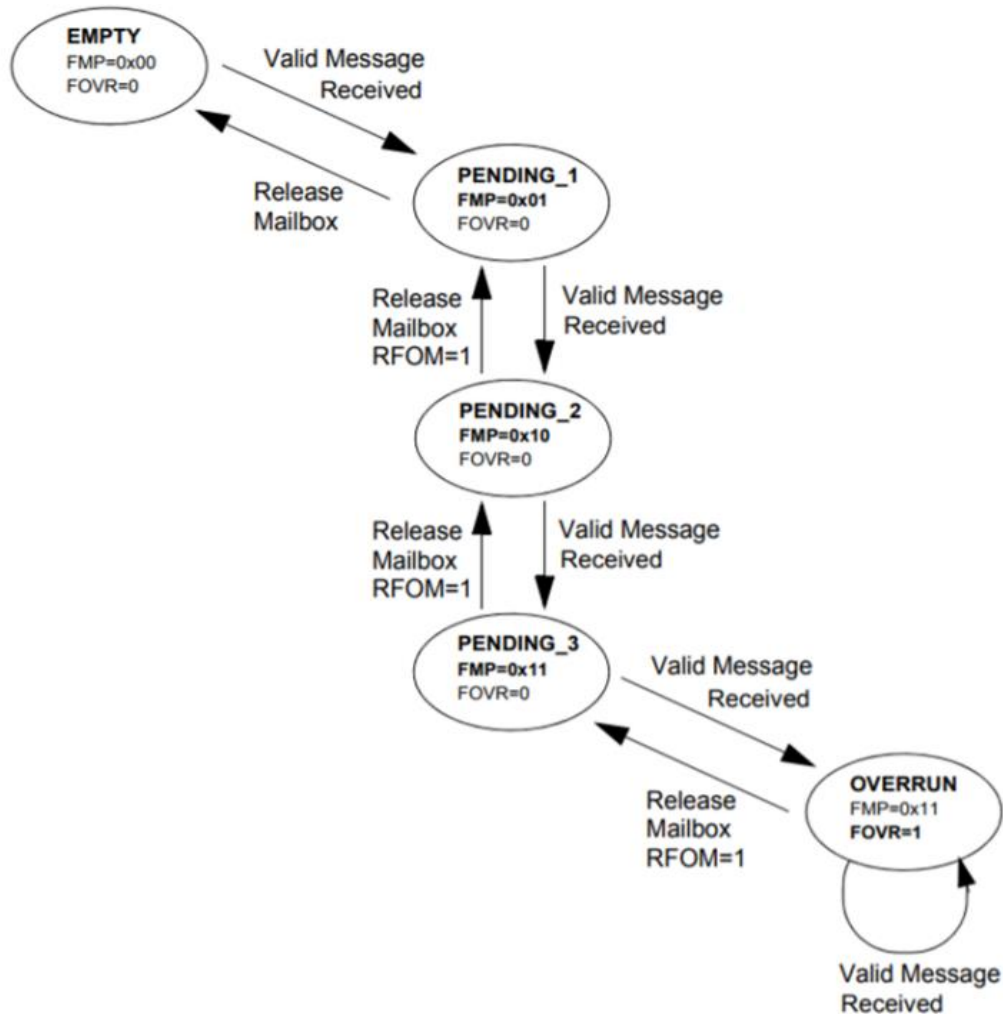
CAN mesajlarının alımı için FIFO olarak düzenlenmiş üç posta kutusu sağlanmıştır.

- CPU yükünden tasarruf etmek, yazılımı basitleştirmek ve veri tutarlılığını garanti etmek için FIFO

tamamen donanım tarafından yönetilir.

- Uygulama, FIFO'da depolanan mesajlara FIFO çıkış posta kutusu aracılığıyla erişilir.
- Alınan bir mesaj, CAN protokolüne göre doğru bir şekilde alındığında geçerli sayılır (EOF alanının

sonuncusuna kadar bir hata olmaz) ve tanımlayıcı filtrelemeden başarıyla geçer.



**Şekil 6 : Receive FIFO States**

**FIFO YÖNETİMİ**

Boş durumdan başlayarak, alınan ilk geçerli mesaj FIFO'da PENDING\_1 tarafından saklanır.

- Donanım, CAN\_RFR kaydındaki FMP [1: 0] bitlerinin 01b değerine ayarlanması olayını bildirir.
- Mesaj FIFO çıkış posta kutusunda bulunur.
- Yazılım, posta kutusu içeriğini okur ve CAN\_RFR kayıt defterinde RFOM bitini ayarlayarak serbest bırakır.
- FIFO yeniden boşalır.
- Bu süre zarfında geçerli bir yeni mesaj alındıysa, FIFO PENDING\_1 durumda kalır ve yeni mesaj, çıkış posta kutusunda bulunur.
- Uygulama posta kutusunu serbest bırakmazsa, bir sonraki geçerli mesaj PENDING\_2 durumuna giren FIFO'da saklanır (FMP [1: 0] = 10b).
- FIFO'yu PENDING\_3 durumuna getiren bir sonraki geçerli mesaj için saklama işlemi tekrarlanır (FMP [1: 0] = 11b).
- Bu noktada, yazılım RFOM bitini ayarlayarak çıkış posta kutusunu serbest bırakmalıdır, böylece bir posta kutusu bir sonraki geçerli mesajı saklamak için serbest kalır.
- Aksi halde, bir sonraki geçerli mesaj mesaj kaybına neden olacaktır.

**OVERRUN**

- FIFO PENDING\_3 durumuna geçtiğinde (yani üç posta kutusu dolu) bir sonraki geçerli mesaj alımı aşılmaya neden olacak ve bir mesaj kaybolacaktır.
- Donanım, CAN\_RFR kaydında FOVR bitini ayarlayarak aşırı çalışma durumunu bildirir.
- Hangi mesajın kaybolduğu FIFO'nun yapılandırmasına bağlıdır:
  - > FIFO kilit işlevi devre dışı bırakılmışsa (CAN\_MCR kaydındaki RFLM biti silindi), FIFO'da depolanan son mesaj yeni gelen mesajın üzerine yazılır. Bu durumda en son mesajlar her zaman uygulamaya açık olacaktır.
  - > FIFO kilit işlevi etkinse (CAN\_MCR kaydında RFLM biti ayarlanmışsa) en son mesaj atılır ve yazılım FIFO'daki en eski üç mesaja sahip olur

**MESAJ ALIM KESMELERİ**

- Program yeni bir mesajın geldiğini nasıl biliyor?
- Mesaj FIFO'ya kaydedildikten sonra, FMP [1: 0] bitleri güncellenir ve bir kesme isteği oluşturulur (CAN\_IER kaydındaki FFIE biti ayarlanmışsa).
- Üç posta kutusunun tamamı doldurulduğunda, CAN\_IER kaydındaki TAM bit ayarlanır.
- Bir aşırı yükleme sırasında FOVR bitinin ayarlandığına ve bir kesme işleminin yalnızca CAN\_IER kaydının FOVIE biti ayarlanmışsa kesme üretileceğine dikkat edilmelidir.
- Aksi takdirde, bir aşırı yükleme sırasındaki tüm yeni mesajlar tamamen göz ardı edilir ve en az bir posta kutusu serbest bırakılıncaya kadar alımlarından dolayı kesinti olmaz.

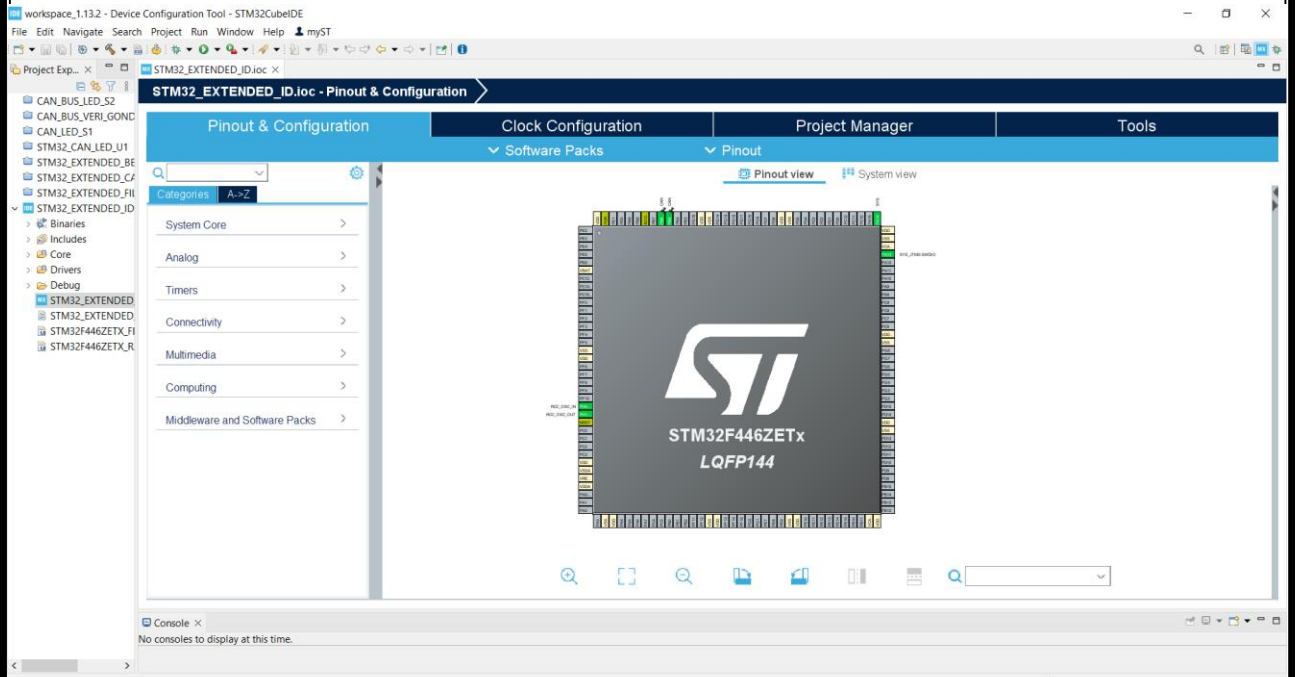
**CAN BUS FİLTRE VE MASKE**

- CAN BUS birimi, istenmeyen mesajları filtrelemek için kabul fitresi ve maske değerini kullanarak bu görevi yerine getirmek için ürün yazılımı içerir.
- Filtre maskesi, alınan çerçevenin tanımlayıcısındaki hangi bitleri karşılaştıracakını belirlemek için kullanılır.
  - > Bir maske biti sıfıra ayarlanmışsa (0x0000), gelen ID bit, filtre bitinden bağımsız olarak otomatik olarak alınır.
  - > Bir maske biti bire ayarlanmışsa (0xFFFF), karşılık gelen ID biti, filtre biti ile karşılaştırılır. Eşleşme olursa kabul edilir, aksi takdirde çerçeve reddedilir.
- Örneğin yalnızca 00001234 kimliği içeren çerçeveleri almak istiyorsak maskeyi 1FFFFFFF olarak ayarlamalıyız.
  - Filtre: 0x00001234
  - Maske 0x1FFFFFFF
- > Bir çerçeve geldiğinde ID'si filtre ile karşılaştırılır ve tüm bitlerin eşleşmesi gerekir (yani tüm bitler sırası ile tek tek karşılaştırılır). 00001234 kimliği ile eşleşmeyen herhangi bir çerçeve reddedilir.

Çalışılan Kısım: AKİBA AR-GE	Yapılan İş: Standart ID – Extended ID
<p>CAN (Controller Area Network) protokolünde, mesajların tanımlanması için kullanılan iki tür kimlik (ID) vardır: Standard ID (Standart Kimlik) ve Extended ID (Genişletilmiş Kimlik).</p> <p><b>Standard ID (Standart Kimlik)</b>  Uzunluk: 11 bit.  Kullanım: Standart kimlik, daha az bit sayısı ile daha kısa mesaj kimliği gerektiren uygulamalarda kullanılır.  Örnek: 0x7FF gibi bir değer alabilir.  Avantajları: Daha düşük veri yükü ve daha hızlı işlem süresi sağlar, özellikle yüksek veri hızına ihtiyaç duyulan uygulamalarda tercih edilir.</p> <p><b>Extended ID (Genişletilmiş Kimlik)</b>  Uzunluk: 29 bit.  Kullanım: Daha uzun kimlikler gerektiren, yani daha fazla mesaj tipini desteklemesi gereken sistemlerde kullanılır.  Örnek: 0x1FFFFFFF gibi bir değer alabilir.  Avantajları: Daha fazla farklı mesajı tanımlama olanağı sağlar. Bu, karmaşık sistemlerde veya çok sayıda cihazın birbirine bağlı olduğu ağlarda faydalıdır.</p> <p><b>Standart ID ile Extended ID Arasındaki Farklar</b>  Mesaj Kimliği Uzunluğu: Standart ID 11 bit, Extended ID ise 29 bit uzunluğundadır.  Veri Yoğunluğu: Extended ID daha fazla bit içerdiği için, iletim sırasında daha fazla veri taşır, bu da hafif bir gecikmeye neden olabilir.  Kullanım Senaryoları: Standart ID genellikle daha basit ağ yapılarında ve daha az cihazın bulunduğu sistemlerde kullanılırken, Extended ID daha büyük ve karmaşık ağ yapılarında tercih edilir.</p>	
Kontrol Eden:	Tarih: 23/07/2024
	Sayfa No 11

**STM32Fxx Serisi Kartları Kullanarak CAN-BUS Nasıl Kullanılır?**

İlk olarak, STM32CubeIDE'yi bilgisayarınıza kurmanız gerekmektedir. Kurulum tamamlandıktan sonra, yeni bir proje oluşturup, kullanacağınız STM32 kart modelini seçmelisiniz. Kart modelini girdikten sonra, karşınıza STM32 mikrodenetleyicisinin pin bacaklarının yer aldığı bir arayüz çıkacaktır. Bu arayüzde, kullanmak istediğiniz özellikleri ve pinleri kolayca yapılandırabilirsiniz.

**Şekil 7: CubeIDE Arayüz**

Kartımızın özelliklerine bağlı olarak, CAN1 ve CAN2 gibi farklı CAN hatları olabilir. Kullanmak istediğiniz CAN hattını aktif hale getirmelisiniz. Ardından, Clock Configuration (Saat Yapılandırması) oldukça önemlidir; kartınızın özelliklerine göre uygun bir saat yapılandırması yapmalısınız.

Bu aşamada, System Core sekmesinden RCC ayarlarını açın ve gerekli ayarları yapın. Örneğin, HSE (High-Speed External) osilatörünü kullanarak PLLCLK'yı ayarlayabilirsiniz. Bu uygulamada, HSE frekansı 8 MHz olarak ayarlanmalıdır.

Daha sonra, CAN hattınıza gidip bu hattı aktif hale getirin. Göndereceğiniz CAN verisinin doğru bir şekilde iletilmesi için, her iki tarafın (STM32 kartı ve diğer modül gibi) baudrate (veri iletim hızı) değerlerinin eşit olması gerekmektedir. Eğer bu değerler eşit değilse, CAN verisi iletilmez.

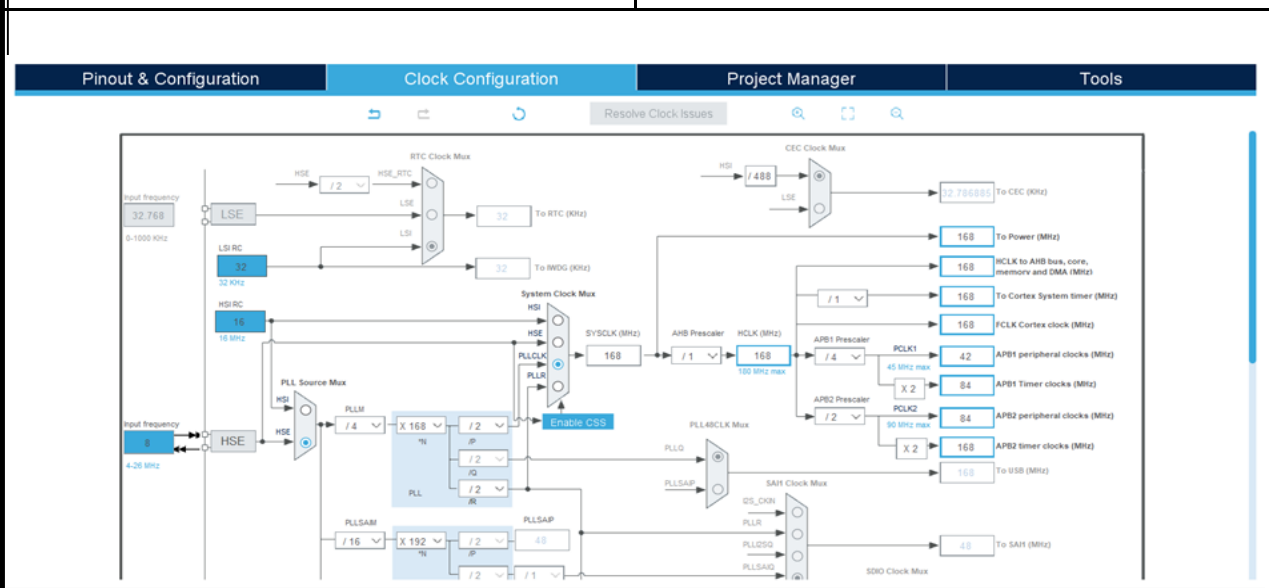
Kontrol Eden:

Tarih:

24/07/2024

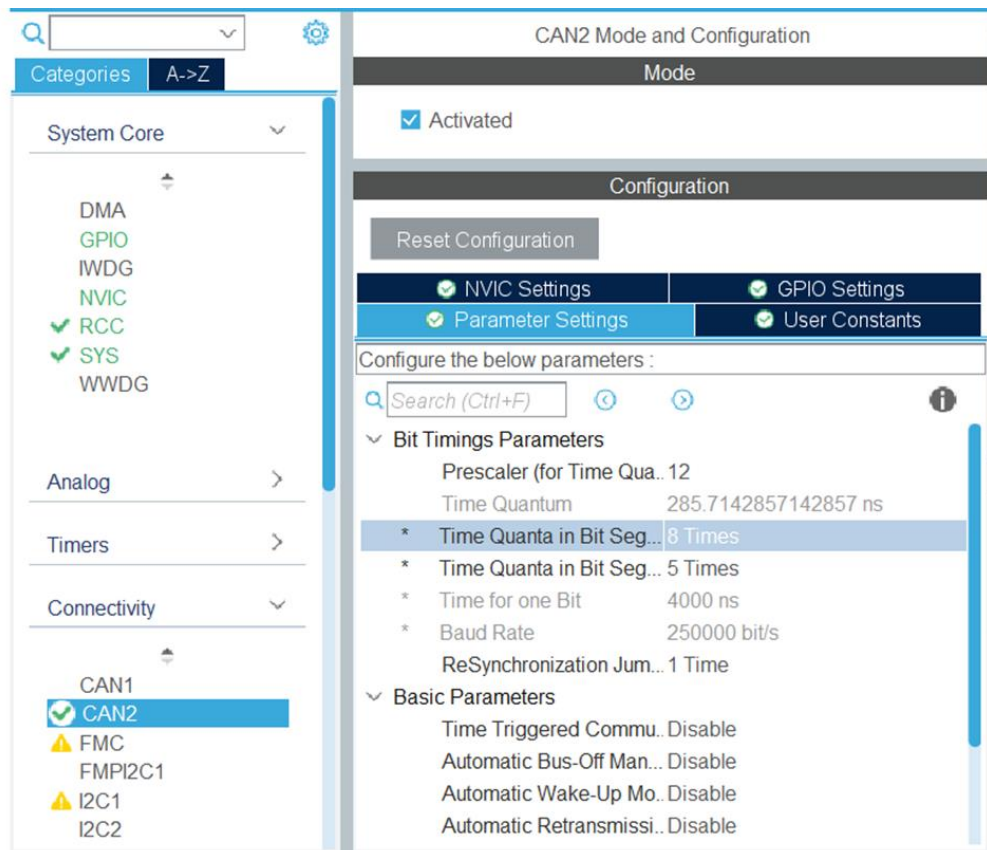
Sayfa No

12



### Şekil 8: Clock Configuration

CAN parametreleri arasında CAN hızını ve diğer birçok ayarı yapabilirsiniz. Bu örnekte, yalnızca CAN hızını ayarladım.



### Şekil 9 : Baud Rote Ayarı



Bu ayardan sonra kod kısmına geçebiliriz. Kod kısmında, CAN iletişimi için bazı parametreleri tanımlamamız gerekiyor.

**CAN\_TxHeaderTypeDef:** CAN veri gönderimi için gerekli başlık (header) yapılandırmasını tutan veri tipidir. İçinde gönderilecek mesajın kimliği (ID), veri uzunluğu, veri türü gibi bilgiler bulunur.

**CAN\_RxHeaderTypeDef:** Gelen CAN mesajları için başlık (header) yapılandırmasını tutan veri tipidir. Bu yapı, alınan mesajın kimliği, veri uzunluğu gibi bilgileri içerir.

**CAN\_FilterTypeDef:** CAN mesajlarını filtrelemek için kullanılan bir yapılandırma tipidir. Bu yapı ile belirli ID'lere sahip mesajların kabul edilmesini veya reddedilmesini sağlayabilirsiniz.

**unsigned char TxData[8]:** Bu dizi, CAN hattı üzerinden gönderilecek veri bytesını içerir. TxData adlı bu dizi, 8 byte uzunluğunda veri taşıyabilir ve bu veriler CAN mesajının içeriğini oluşturur.

**unsigned char RxData[8]:** Bu dizi, CAN hattı üzerinden alınan veri bytesını içerir. RxData adlı bu dizi, 8 byte uzunluğunda veri alabilir ve gelen CAN mesajının içeriğini saklar.

**uint32\_t TxMailbox:** Bu değişken, CAN mesajlarının gönderimi sırasında kullanılan iletim posta kutusunun (mailbox) tanımlayıcısını içerir. CAN verisi gönderilirken, hangi posta kutusunun kullanıldığını belirler ve verinin gönderilmesini yönetir.

**HAL\_CAN\_Start(&hcan1);** fonksiyonu, CAN modülünü başlatır ve CAN iletişim hattında veri iletimine izin verir. Bu adım tamamlandıktan sonra, CAN filtrelerini tanımlamak ve yapılandırmak için gerekli işlemleri yapabilirsiniz.

Filtreler, belirli ID'lere sahip CAN mesajlarını almak için kullanılır ve ağ üzerindeki gereksiz mesajları filtrelemeye yardımcı olur. Aşağıda, filtrelerin nasıl tanımlanacağına ve yapılandırılacağına dair kısa bir açıklama verilmiştir:

Filtre Yapılandırması:

Filtreler, CAN\_FilterTypeDef yapısı kullanılarak tanımlanır. Bu yapı, filtreleme kriterlerini belirlemenize olanak tanır. Tipik olarak, bu yapı şu alanları içerir:

**FilterIdHigh:** Filtre ID'sinin yüksek kısmı.

**FilterIdLow:** Filtre ID'sinin düşük kısmı.

**FilterMaskIdHigh:** Filtre maskesinin yüksek kısmı.

**FilterMaskIdLow:** Filtre maskesinin düşük kısmı.

**FilterFIFOAssignment:** Filtreleme için kullanılacak FIFO'yu belirtir.

**FilterMode:** Filtreleme modunu (standart veya genişletilmiş ID) belirler.

**FilterScale:** Filtreleme ölçeğini (tek veya çift 32-bit) belirler.

**FilterActivation:** Filtrenin aktif olup olmadığını belirtir.

Bu filtre yapılandırması veri alacak olan kart içindir.

Kişi veri gönderecek veya alacak olduğunda ayarlamalarını bu tanımlar ışığında gerçekleştirmelidir

Çalışılan Kısım: AKİBA AR-GE	Yapılan İş: CAN-BUS Uygulama
<p>CAN verisi gönderme işlemini yapılandırmak için CAN_TxHeaderTypeDef yapısını kullanırız. İşte bu yapı ile ilgili açıklamalar ve örnek kullanım:</p> <p><b>CAN_TxHeaderTypeDef Yapısı</b> Bu yapı, CAN mesajının başlık bilgilerini içerir. Gönderim işlemi için gerekli olan bazı alanlar:</p> <p><b>StdId:</b> Gönderilecek CAN mesajının standart ID'sini belirtir. ID, mesajın ağ üzerindeki benzersiz tanımlayıcısıdır. Örneğin, 0x601 bir ID değeri olarak kullanılabilir.</p> <p><b>DLC:</b> "Data Length Code" (Veri Uzunluk Kodu) olarak bilinir. Bu alan, mesajın veri uzunluğunu belirtir. Bir CAN mesajı, 0'dan 8 byte'a kadar veri içerebilir. Örneğin, 8 burada mesajın 8 byte veri içerdiğini belirtir.</p> <p><b>IDE:</b> "Identifier Extension" (Kimlik Uzantısı) olarak bilinir. Bu alan, ID'nin standart (11-bit) veya genişletilmiş (29-bit) olup olmadığını belirtir. CAN_ID_STD standart ID'yi kullanır.</p> <p><b>RTR:</b> "Remote Transmission Request" (Uzak İletim Talebi) olarak bilinir. Bu alan, mesajın veri gönderme (data) veya talep etme (remote) olduğunu belirtir. CAN_RTR_DATA veri gönderme modunu belirtir.</p> <p><b>Veri Gönderme (CAN_Mesajı Gönderme)</b> Fonksiyon: HAL_CAN_AddTxMessage Parametreler: hcan: CAN_HandleTypeDef tipinde, CAN periferik yapılandırma bilgilerini içeren yapı. Bu, CAN modülünü tanımlar. pHeader: CAN_TxHeaderTypeDef tipinde, gönderilecek CAN mesajının başlık bilgilerini içeren yapı. Bu, mesajın kimliği, uzunluğu ve diğer başlık bilgilerini içerir. pData: uint8_t dizisi, gönderilecek veri. Maksimum 8 byte olabilir. pTxMailbox: uint32_t tipinde, gönderim için kullanılacak FIFO kuyruğunun numarasını belirten işaretçi. Gönderim tamamlandığında, bu değişken FIFO kuyruğunun numarasını içerir.</p> <p><b>Veri Alma (CAN_Mesajı Alma)</b> Fonksiyon: HAL_CAN_GetRxMessage Parametreler: hcan: CAN_HandleTypeDef tipinde, CAN periferik yapılandırma bilgilerini içeren yapı. Bu, CAN modülünü tanımlar. FIFO: uint32_t tipinde, mesajın alındığı FIFO kuyruğu numarası. CAN_RX_FIFO0 veya CAN_RX_FIFO1 gibi değerler alabilir. pHeader: CAN_RxHeaderTypeDef tipinde, alınan CAN mesajının başlık bilgilerini içerecek yapı. Bu, mesajın kimliği, uzunluğu ve diğer başlık bilgilerini içerir. pData: uint8_t dizisi, alınan veri. Maksimum 8 byte olabilir.</p>	
Kontrol Eden:	Tarih: 29/07/2024
	Sayfa No 15

**KOD**

```

/* Private user code -----*/
/* USER CODE BEGIN 0 */
CAN_TxHeaderTypeDef TxTransmitter;
CAN_RxHeaderTypeDef RxReceiver;

unsigned char TxData[8];
unsigned char RxData[8];
uint32_t TxMailbox;
/* USER CODE END 0 */

```

Şekil 10: Kod 1

```

MX_GPIO_Init();
MX_CAN2_Init();
MX_CAN1_Init();
/* USER CODE BEGIN 2 */
HAL_CAN_Start(&hcan1);
HAL_CAN_Start(&hcan2);
//HAL_CAN_ConfigFilter(&hcan1, &can_filter);
//HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
/* Start the CAN peripheral */

/* Activate CAN RX notification */
if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
{
    Error_Handler();
}
sFilterConfig.FilterBank = 0;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x0000;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x0000;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
sFilterConfig.FilterActivation = ENABLE;
if (HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig) != HAL_OK)
{
    Error_Handler();
}

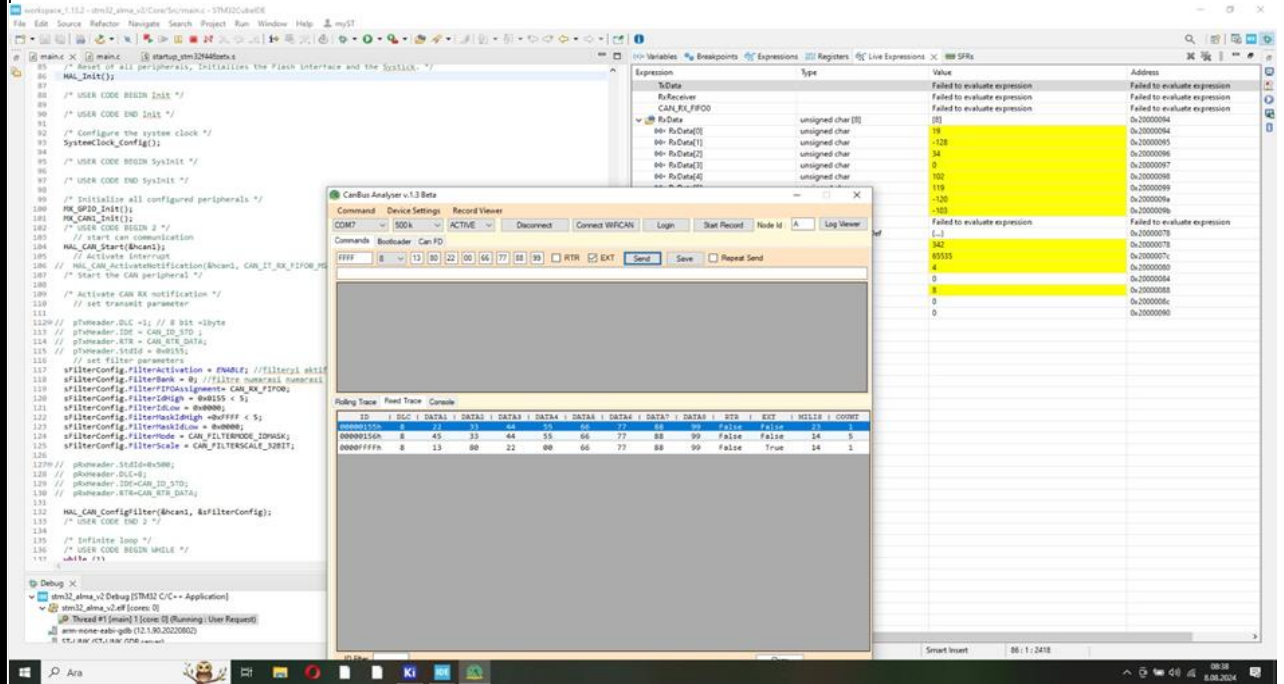
TxTransmitter.StdId=0x601;
TxTransmitter.DLC=8;
TxTransmitter.IDE=CAN_ID_STD;
TxTransmitter.RTR = CAN_RTR_DATA;

```

Şekil 11: Kod 2

```
TxData[0] = 0x60;  
TxData[1] = 0x0;  
TxData[2] = 0x6;  
TxData[3] = 0x51;  
TxData[4] = 0x21;  
TxData[5] = 0x11;  
TxData[6] = 0x34;  
TxData[7] = 0x01; //sensor_time * .034 /2;  
  
RxReceiver.StdId=0x581;  
RxReceiver.DLC=8;  
RxReceiver.IDE=CAN_ID_STD;  
RxReceiver.RTR=CAN_RTR_DATA;  
  
/* USER CODE END 2 */  
  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    /* USER CODE END WHILE */  
  
    /* USER CODE BEGIN 3 */  
  
    //HAL_CAN_AddTxMessage(&hcan1, &TxTransmitter, TxData, &TxMailbox);  
  
    // HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0 , &RxReceiver, RxData);  
    if (HAL_CAN_AddTxMessage(&hcan1, &TxTransmitter, TxData, &TxMailbox) != HAL_OK)  
    {  
        Error_Handler();  
    }  
}
```

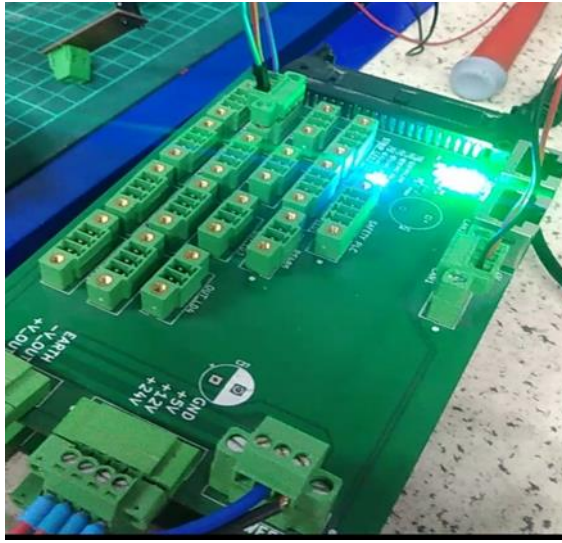
Şekil 12: KOD 3



Şekil 13: Uygulama Çıktısı

**CAN BUS LED UYGULAMASI**

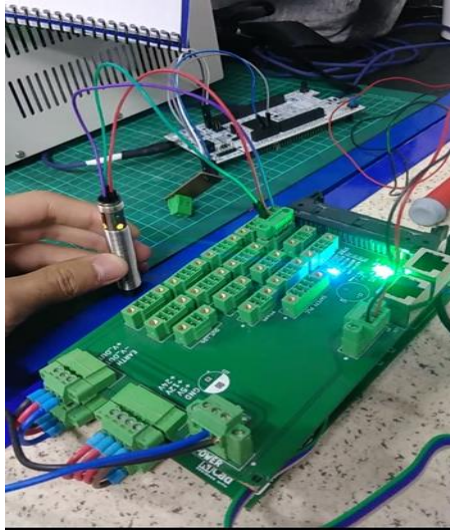
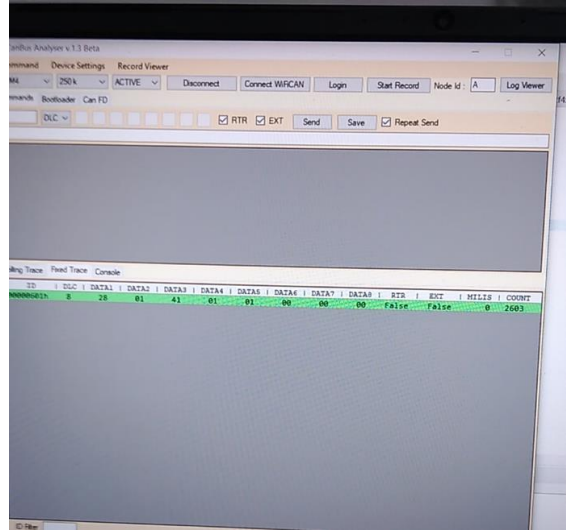
STM32F446ZETx ile CAN-Bus Analyser kullanarak gerçekleştirilecek bir uygulamada, STM32 mikrodenetleyicisi üzerinden 601 Frame ID değerine sahip bir CAN mesajı gönderilecektir. Gönderilen bu mesajın 8. byte'ı 0x01 değerine eşit olduğunda, STM32 üzerindeki dahili LED'lerden biri 250ms periyotla yanıp sönecektir. Eğer 8. byte 0x01 ve 7. byte 0x00 değerine sahipse, aynı LED 1 saniyelik periyotla yanıp sönecek şekilde ayarlanacaktır.

**Şekil 14:** Led Toggle

**CAN\_BUS\_ENDÜKTİF\_SENSÖR**

Bu projede, endüktif sensörün RECU üzerindeki DIN4 portuna bağlanarak STM32 mikrodenetleyicisinin PE13 pinine dijital giriş olarak tanımlanması sağlanmıştır. Sensörden alınan dijital sinyal, bu pin üzerinde tutulmaktadır. CAN-BUS sürekli olarak dışarıya veri göndermekte olup, varsayılan durumda gönderilen veri şu şekildedir: Node ID = 601, DATA = 40 01 65 01 00 00 00.

Eğer endüktif sensör bir metal bileşen algıarsa, CAN-BUS hattındaki verinin 5. byte'ı 0x01 olarak güncellenecektir. Bu güncelleme, sensör tarafından metal algılandığı sürece devam edecektir. Veriler, CAN-BUS üzerinden 100ms periyodik aralıklarla iletilecektir ve CAN-BUS Delacon arayüzü kullanılarak 250kbps hızında izlenecektir.

**Şekil 15 : Sensör ve led görüntüsü****Şekil 16: Can Analayser görüntüsü**

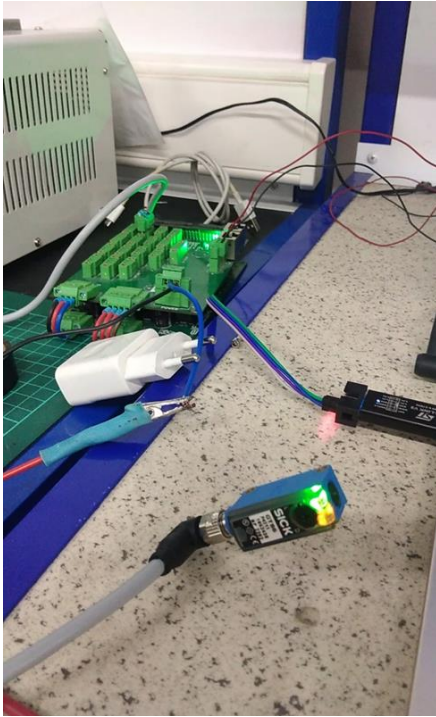
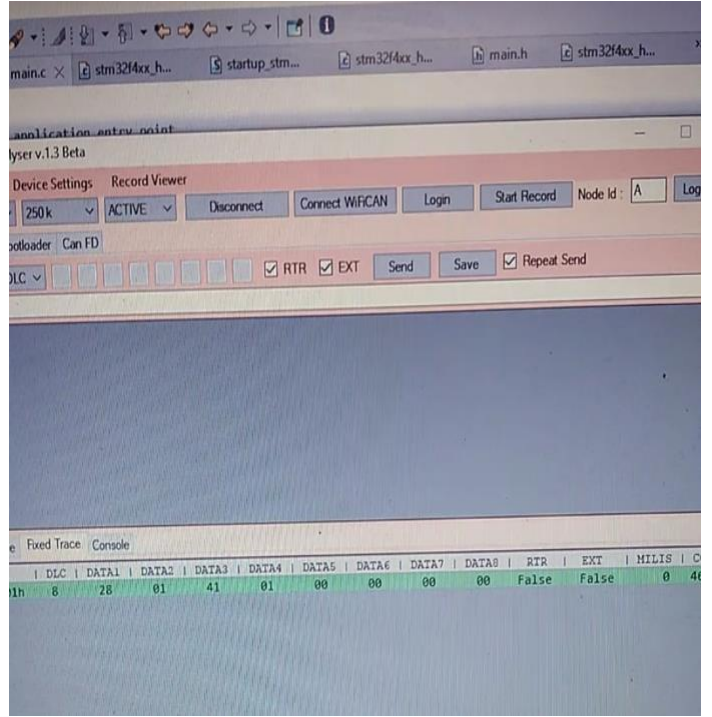


**CAN\_BUS GTB6\_LASER\_SENSÖR**

Bu projede, STM32 mikrodnetleyicisi kullanılarak GTB6\_LASER\_SENSÖR'ün 0 ile 24V aralığındaki dijital giriş sinyali okunmaktadır. Okunan bu sinyal, dijital giriş olarak mikrodnetleyicinin ilgili pinine atanmıştır.

Projede, sürekli olarak CAN-BUS üzerinden 601 ID numaralı 8 byte'lık veri akışı bulunmaktadır. Sensörden herhangi bir input geldiğinde, STM32 üzerindeki kullanıcı LED'i yakılacak ve aynı zamanda CAN-BUS üzerinden gönderilen verinin 5. byte'ının son bit değeri 0x01 olarak güncellenecektir.

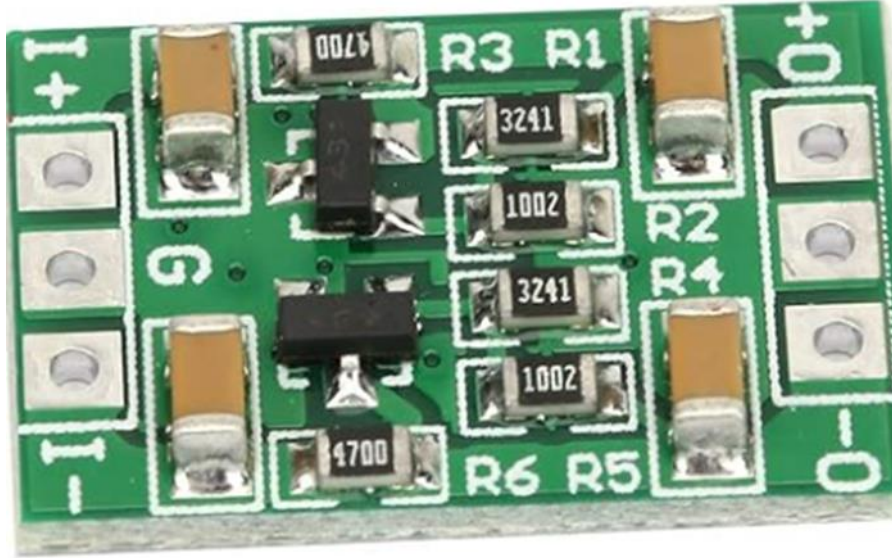
Bu işlemler, GTB6\_LASER\_SENSÖR'den gelen sinyale bağlı olarak gerçek zamanlı olarak yapılacaktır ve veriler CAN-BUS Analayser ile izlenebilecektir.

**Şekil 17: SICK Sensörü****Şekil 18 : CAN Analayser Çıktısı**



**STM32F446ZET KARTINA LEHİM UYGULAMASI**

Şirkette bulunan üç adet STM32F446ZET mikrodenetleyici kartına ADC haberleşmesi için gerekli modüllerin lehimleme işlemini gerçekleştirdim. Bu süreçte, her bir kart üzerindeki modüllerin güvenli ve işlevsel bir şekilde yerleştirilmesi için büyük özen gösterdim. Kartların stabil çalışmasını sağlamak adına, lehim bağlantılarının doğru ve sağlam olmasına dikkat ettim. Ayrıca, modüllerin daha sonraki test ve kullanım aşamalarında sorunsuz çalışabilmesi için tüm lehim noktalarını titizlikle kontrol ettim. Bu işlemler, kartların ADC haberleşmesi gerektiren endüstriyel projelerde kullanılmak üzere hazır hale getirilmesini sağlamıştır.

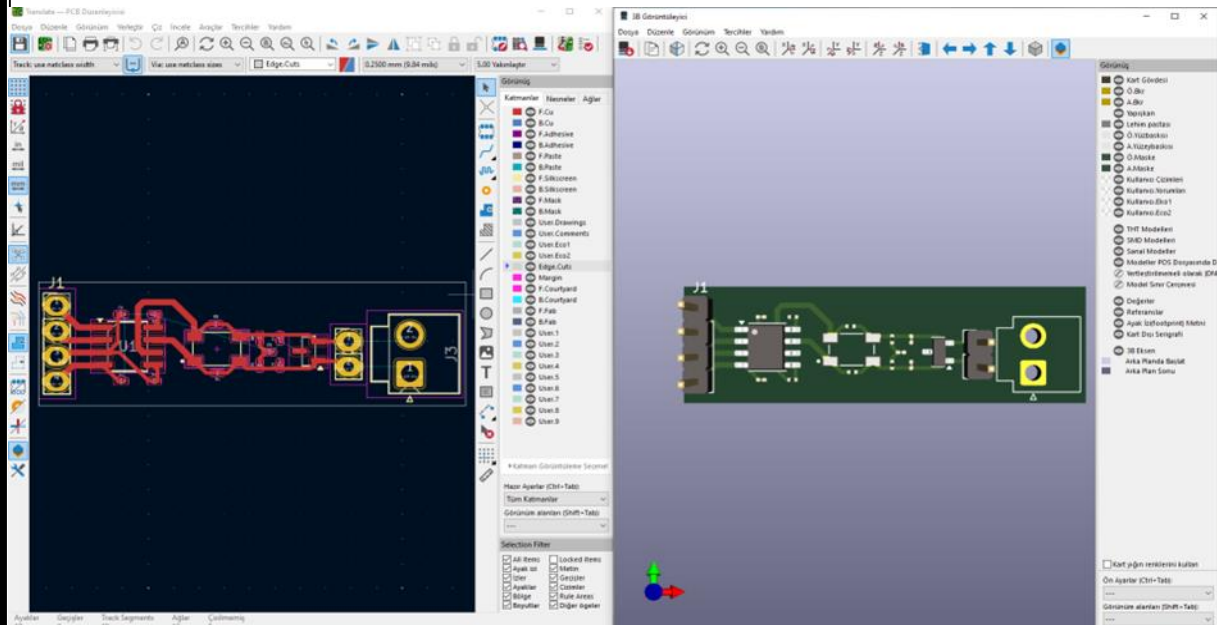
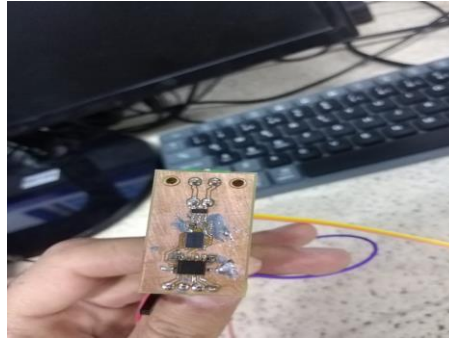


**Şekil 19:** ADC Elektronik Yapısı

**KiCad Uygulaması ile CAN Modülü Tasarımı**

Staj dönemimde, KICAD yazılımını kullanarak bir CAN BUS modülü tasarladım ve bu tasarımın lehimleme işlemlerini gerçekleştirdim. İlk olarak, CAN BUS modülünün devre şemasını ve PCB tasarımını KICAD üzerinde detaylı bir şekilde oluşturdum. Bu süreçte, devre elemanlarının doğru yerleştirilmesi, izlerin uygun şekilde yönlendirilmesi ve modülün optimal performans gösterebilmesi için gerekli tüm tasarım kurallarına özen gösterdim.

Tasarım tamamlandıktan sonra, PCB üretim sürecinin ardından elime ulaşan kartların üzerinde lehimleme işlemlerini gerçekleştirdim. Her bir bileşeni dikkatle yerleştirip lehimleyerek, modülün güvenli ve işlevsel olmasını sağladım. Son olarak, modülün tüm bağlantı noktalarını titizlikle kontrol ederek, olası lehim hatalarını giderdim ve modülün sorunsuz bir şekilde çalışabilmesi için gerekli testleri gerçekleştirdim. Bu çalışma, staj sürecimde edindiğim teknik bilgi ve becerileri pratikte uygulama fırsatı sundu.

**Şekil 20: KiCAD Şematik ve 3D Görünüm****Şekil 21 : Kartın Son Hali**

Kontrol Eden:

Tarih:  
08/08/2024Sayfa No  
23

Çalışılan Kısım: AKİBA AR-GE

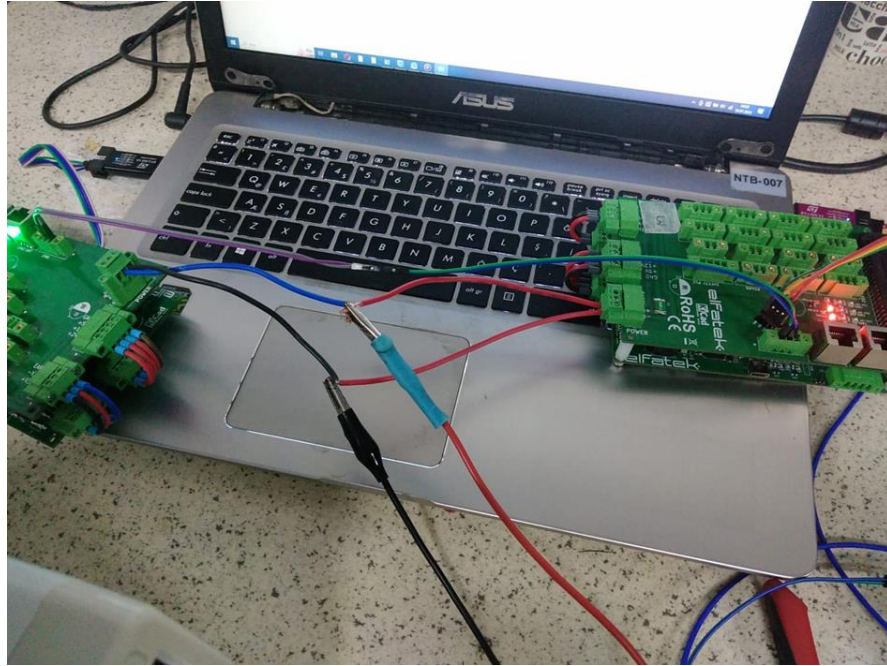
Yapılan İş: STM32F446ZET Kartlar Arası CAN Haberleşme

### STM32F446ZET Kullanarak Extended id ile Haberleşme

Staj sürecimde, STM32F446ZET mikrodenetleyicisini kullanarak Extended ID ile CAN-BUS haberleşmesi uygulaması gerçekleştirdim. Bu uygulama, özellikle daha karmaşık ve büyük veri setlerinin yönetilmesi gereken sistemlerde kullanılan 29-bitlik Extended ID'leri temel alıyordu.

Proje kapsamında, STM32F446ZET kartı üzerinde CAN-BUS protokolünü yapılandırarak, farklı ID'lere sahip mesajların doğru bir şekilde gönderilmesi ve alınmasını sağladım. Extended ID'lerin avantajlarından yararlanarak, daha fazla mesaj çeşidinin aynı ağ üzerinden iletilmesine olanak tanıyan bu uygulama, veri trafiğinin yönetimi ve sistemlerin entegrasyonu açısından önemli bir adım oldu.

Uygulama sürecinde, CAN-BUS konfigürasyonları, filtreleme mekanizmaları ve veri gönderme-alma işlemleri gibi temel CAN protokol özelliklerini detaylı bir şekilde ele aldım. Geliştirdiğim bu sistem, araç içi ağlar veya endüstriyel otomasyon sistemleri gibi birçok farklı alanda kullanılabilecek nitelikte olup, daha büyük ve karmaşık sistemlerin haberleşme gereksinimlerini karşılayabilme kapasitesine sahiptir.



Şekil 21: İki Farklı STM32 Haberleşme Görüntüsü

Kontrol Eden:

Tarih:  
09/08/2024

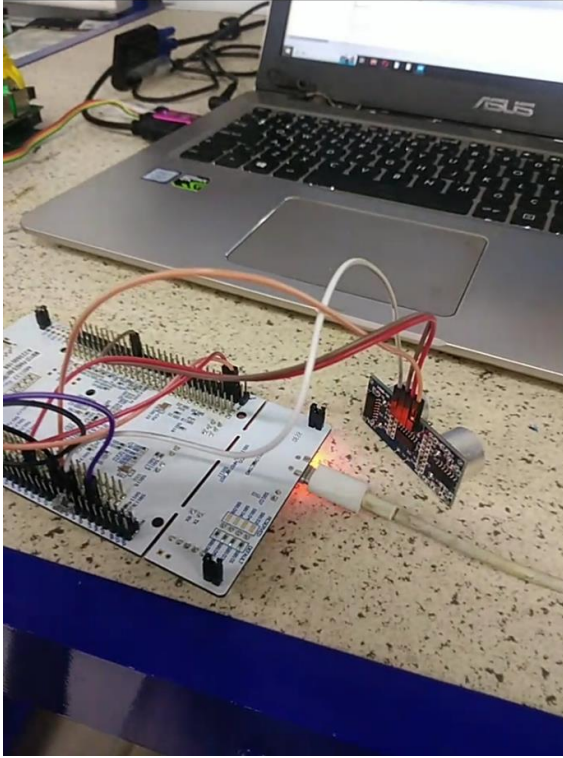
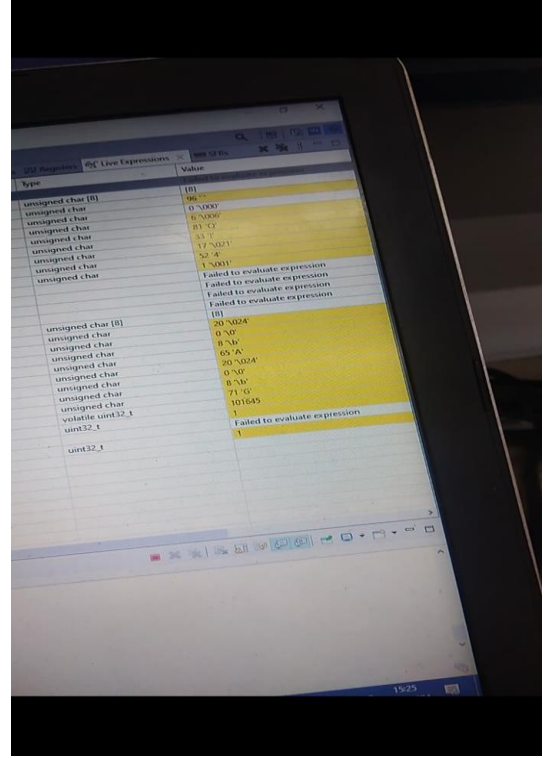
Sayfa No  
24

**STM32F446ZET ile HC-SR04 Verisini CAN ile Gönderme**


Staj sürecimde, STM32F446Zxx mikrodnetleyicisini kullanarak HC-SR04 ultrasonik sensöründen mesafe verisi okuma ve bu veriyi CAN-BUS hattı üzerinden iletme üzerine bir uygulama geliştirdim.

Bu projede, HC-SR04 sensörünü STM32F446Zxx mikrodnetleyiciye bağlayarak, sensörden gelen yankı sinyalini kullanarak mesafe ölçümlerini gerçekleştirdim. Elde edilen mesafe verileri, mikrodnetleyici tarafından işlendikten sonra CAN-BUS hattı üzerinden aktarılmak üzere hazırlandı.

CAN-BUS hattı üzerinden iletilecek verilerin, belirlenen standart ID altında doğru bir formatta iletilmesini sağladım. Bu sayede, diğer CAN-BUS uyumlu cihazlar bu veriyi okuyabilir ve uygun aksiyonları alabilir. Örneğin, bir araç içi sistemde bu veri, engel algılama veya park yardımı gibi uygulamalarda kullanılabilir.

**Şekil 22: HC-SR04 Bağlantı****Şekil 23: CAN Mesaj Bilgisi**



Çalışılan Kısım: AKİBA AR-GE	Yapılan İş: DMU330 Mesafe Sensörünü Okuma ve CAN BUS ile Veri Gönderme
<p><b>LEUZE DMU330 Sensör</b></p> <p>Staj sürecimde, endüstriyel LEUZE DMU330 mesafe sensörünü kullanarak STM32F446ZET mikrodnetleyicisi ile bir uygulama geliştirdim. Bu projede, LEUZE DMU330 sensöründen alınan mesafe verilerini ADC (Analog-Dijital Çevirici) üzerinden okuyarak işledim. Ardından, bu ölçülen mesafe değerini CAN-BUS hattı üzerinden başka bir STM32 mikrodnetleyiciye ilettim.</p> <p>Bu haberleşme işlemi sırasında, CAN-BUS protokolünde <i>Extended ID</i> (Genişletilmiş Kimlik) kullanılarak, daha fazla veri taşıma kapasitesi ve daha karmaşık ağ yapılarına uyum sağlandı. Bu yaklaşım, özellikle endüstriyel otomasyon sistemlerinde, birden fazla cihazın aynı anda veri alışverişi yapması gereken durumlar için kritik öneme sahiptir.</p> <p>Sonuç olarak, bu proje ile endüstriyel bir sensörden alınan verilerin, güvenilir bir şekilde başka bir mikrodnetleyiciye iletilmesi sağlandı ve CAN-BUS hattı üzerinden genişletilmiş ID kullanılarak iletişim gerçekleştirildi. Bu uygulama, endüstriyel otomasyon sistemlerinde sensör verilerinin toplanması ve işlenmesi konularında önemli bir adım teşkil etmektedir.</p>	
	
<p style="text-align: center;"><b>Şekil 24: DMU330</b></p>	
Kontrol Eden:	<div>Tarih:</div> <div>13/08/2024</div>
	<div>Sayfa No</div> <div>26</div>

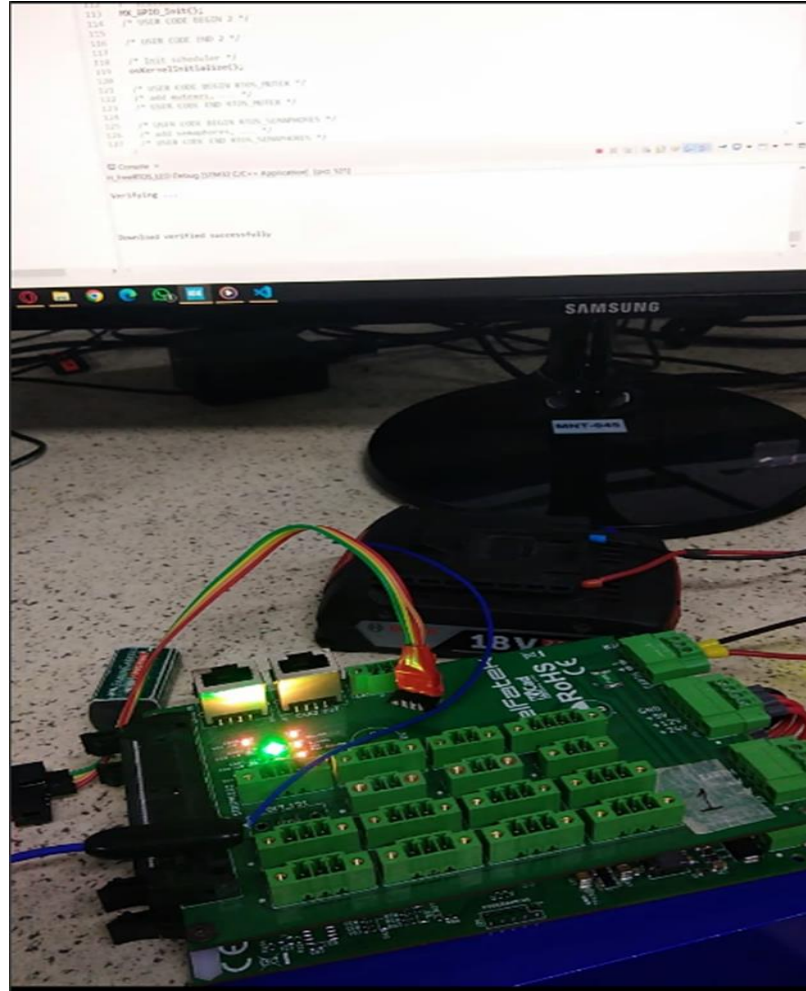
Çalışılan Kısım: AKİBA AR-GE	Yapılan İş: FreeRTOS Öğrenme	
<p><b>FreeRTOS Nedir?</b></p> <p>FreeRTOS, açık kaynaklı ve hafif bir gerçek zamanlı işletim sistemi (RTOS) çekirdeğidir. Gömülü sistemler için özel olarak tasarlanmış olup, çoklu görev yürütme, öncelik tabanlı zamanlama, semaforlar, mesaj kuyrukları ve zamanlayıcılar gibi temel işletim sistemi işlevlerini sunar. FreeRTOS, esnek ve küçük yapısı sayesinde birçok mikrodenetleyici platformunda, özellikle kaynak kısıtlı sistemlerde yaygın olarak kullanılır.</p> <p><b>FreeRTOS'un Avantajları</b></p> <ul style="list-style-type: none"><li>• Gerçek Zamanlı İşlem: FreeRTOS, belirli görevlerin belirli zaman dilimlerinde veya olaylar meydana geldiğinde çalışmasını sağlar. Bu, zaman kritik uygulamalar için idealdir.</li><li>• Küçük Hafıza Ayak İzi: FreeRTOS, bellek ve işlem gücü açısından oldukça verimlidir, bu da onu düşük kaynaklı mikrodenetleyicilerde kullanım için uygun kılar.</li><li>• Platform Bağımsızlık: FreeRTOS, birçok mikrodenetleyici platformunu destekler. Bu sayede, uygulamayı farklı donanım platformlarına taşıma süreci kolaylaşır.</li><li>• Modüler Yapı: FreeRTOS, sadece ihtiyaç duyulan bileşenlerin kullanılmasıyla modüler bir yapı sunar, bu da gereksiz işlevlerin koddan çıkarılmasını sağlar.</li><li>• Geniş Kullanıcı Topluluğu ve Destek: FreeRTOS, büyük bir kullanıcı topluluğuna sahiptir ve sürekli olarak güncellenir. Ayrıca, dökümantasyonu ve destek kaynakları geniştir.</li></ul> <p><b>FreeRTOS'un Dezavantajları</b></p> <ul style="list-style-type: none"><li>• Zamanlayıcı Sınırlamaları: FreeRTOS, sabit zaman dilimlerine sahip bir zamanlayıcı kullanır, bu da bazı karmaşık zamanlama gereksinimlerinde sınırlı kalabilir.</li><li>• Gerçek Zamanlı Garantilerin Olmaması: FreeRTOS, her ne kadar gerçek zamanlı bir işletim sistemi olarak tasarlanmış olsa da, belirli koşullar altında gerçek zamanlı performans garantisi edilmez. Yüksek yük altında beklenen performansı gösteremeyebilir.</li><li>• Karmaşıklık: RTOS kullanımı, özellikle büyük projelerde, görevler arası iletişim ve zamanlama yönetimi gibi konularda karmaşıklık yaratabilir. Bu, geliştiricinin dikkatli tasarım ve test süreçleri gerektirir.</li></ul>		
Kontrol Eden:	Tarih: 14/08/2024	Sayfa No 27

**FreeRTOS LED Yakma İşlemi**

FreeRTOS kullanarak STM32F446ZET mikrodnetleyici kartımız üzerinde yer alan kullanıcı LED'lerini başarılı bir şekilde kontrol ettik. Bu işlem, FreeRTOS'un temel işlevlerini kullanarak gerçekleştirildi ve LED'lerin aynı anda yanmasını sağladı.

Öncelikle, FreeRTOS içinde birden fazla görev (task) tanımlandı. Bu görevler, her bir LED'i kontrol etmek için tasarlandı. Ardından, FreeRTOS zamanlayıcısı (scheduler) başlatılarak görevlerin belirlenen önceliklere göre aynı anda çalışması sağlandı.

Her görev, ilgili LED'i yakma işlemini gerçekleştirdi ve bu görevler FreeRTOS'un zamanlama ve görev yönetimi yetenekleri sayesinde aynı anda çalıştırıldı. Sonuç olarak, tüm kullanıcı LED'leri aynı anda yanarak FreeRTOS'un çoklu görev yürütme kabiliyetini ve zamanlayıcı yönetimini başarılı bir şekilde gösterdi. Bu proje, FreeRTOS'un STM32F446ZET mikrodnetleyicisi üzerinde nasıl kullanılabileceğini ve FreeRTOS'un gerçek zamanlı kontrol görevlerini nasıl etkin bir şekilde yerine getirebildiğini göstermiş oldu.

**Şekil 25: Ledlerin Yanma Anı**

Kontrol Eden:

Tarih:

15/08/2024

Sayfa No

28

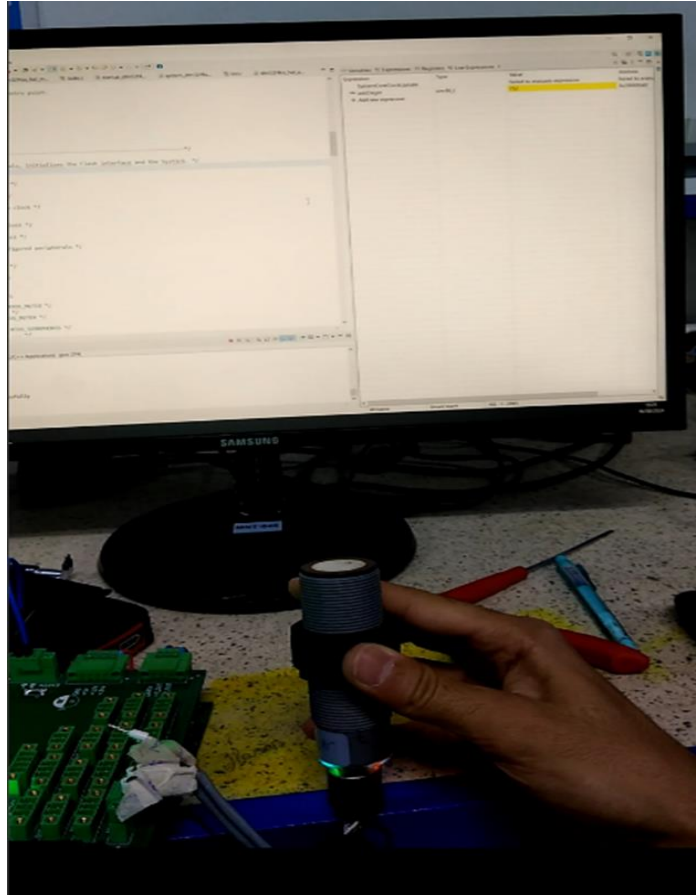
**FreeRTOS ile DMU330 ve Led Projesi**

Bu projede, STM32F446ZET mikrodnetleyici kullanarak FreeRTOS ile bir uygulama geliřtirdik. Bu uygulama, aynı anda hem kullanıcı LED'lerini kontrol etmeyi hem de LEUZE DMU330 mesafe sensöründen veri okumayı içeriyor.

Öncelikle, FreeRTOS'un çoklu görev yönetimi yeteneklerinden faydalanarak iki ana görev oluřturduk: biri LED'leri yakıp söndürmek, diğeri ise sensörden mesafe verilerini okumak. Bu görevler, FreeRTOS zamanlayıcısı tarafından eş zamanlı olarak çalıştırıldı.

LED kontrol görevi, mikrodnetleyici üzerindeki kullanıcı LED'lerinin belirli aralıklarla yanıp sönmesini sağladı. Aynı anda, sensör okuma görevi, LEUZE DMU330 mesafe sensöründen alınan verileri işleyip mikrodnetleyiciye ilette. Bu veriler, daha sonra ekrana yazdırıldı, böylece sensörün algıladıđı mesafe bilgilerini gerçek zamanlı olarak görüntüleyebildik.

Projenin sonucunda, FreeRTOS'un sağladıđı esneklik ve zamanlama yetenekleri sayesinde hem LED'lerin kontrolünü hem de sensör verilerinin okunmasını senkronize bir şekilde gerçekleřtirdik. Bu tür bir uygulama, gerçek zamanlı sistemlerin nasıl yönetileceđine dair pratik bir örnek sundu.



Şekil 26: DMU330 dan FreeRTOS ile okunan değerin görüntüsü



**FreeRTOS ile Okunan Değeri CAN-BUS ile Başka STM32'ye Gönderme Projesi:****Proje: FreeRTOS ile Okunan Değeri CAN-BUS Üzerinden Başka STM32'ye Gönderme**

Bu projede, FreeRTOS kullanarak bir STM32F446ZET mikrodeneleyici üzerinde sensör verisi okuma ve bu veriyi CAN-BUS aracılığıyla başka bir STM32 mikrodeneleyiciye gönderme işlemi gerçekleştirilmiştir. Bu uygulama, FreeRTOS'un çoklu görev yönetimi ve CAN-BUS haberleşme protokolünü kullanarak gömülü sistemler arasında veri iletişimi sağlamayı hedeflemektedir.

**1. FreeRTOS ile Sensör Verisinin Okunması**

Projede ilk adım olarak, FreeRTOS ile bir görev tanımlanarak sensör verisi okunmuştur. Bu örnekte, sensör olarak LEUZE DMU330 mesafe sensörü kullanılmıştır. FreeRTOS üzerinde oluşturulan bir görev, belirli zaman aralıklarıyla sensörden veri okur ve bu veriyi bir değişkene kaydeder.

**2. CAN-BUS Haberleşmesi için Yapılandırma**

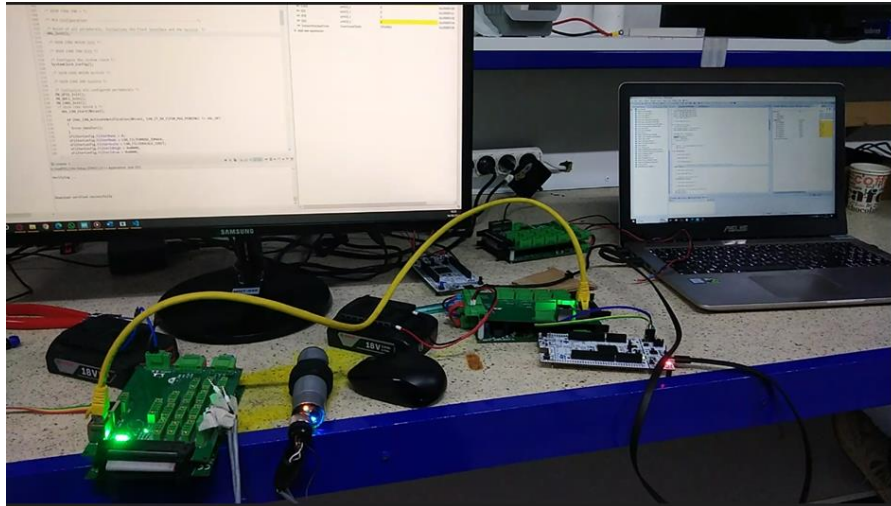
CAN-BUS, endüstriyel ve otomotiv uygulamalarında yaygın olarak kullanılan bir veri iletişim protokolüdür. İki STM32 mikrodeneleyici arasında veri transferi için CAN-BUS kullanılacaktır. STM32CubeMX aracıyla, CAN-BUS modülü yapılandırılır. Bu yapılandırma, CAN hızını, filtre ayarlarını ve gerekli pin tanımlamalarını içerir. CAN-BUS iletişimi için uygun bit hızları ve filtreleme ayarları belirlenir.

**3. Verinin CAN-BUS Üzerinden Gönderilmesi**

Okunan sensör verisi, CAN-BUS göndermeye uygun bir formatta paketlenir. Bu verinin başka bir mikrodeneleyiciye gönderilmesi için HAL\_CAN\_Transmit fonksiyonu kullanılır. CAN çerçevesi, sensör verisini içerir ve başka bir STM32 mikrodeneleyiciye başarılı bir şekilde iletilir.

**4. İkinci STM32'de Verinin Alınması**

Diğer STM32 mikrodeneleyici, CAN-BUS üzerinden gelen veriyi alır ve işler. Bu mikrodeneleyici, aynı zamanda FreeRTOS kullanarak veri alımını bir görev olarak tanımlayabilir ve gelen veriyi işleyecek veya ekrana yazdıracak şekilde yapılandırılabilir.



**Şekil 27: STM32CAN-BUS Haberleşmesi ve Veri Aktarımı**