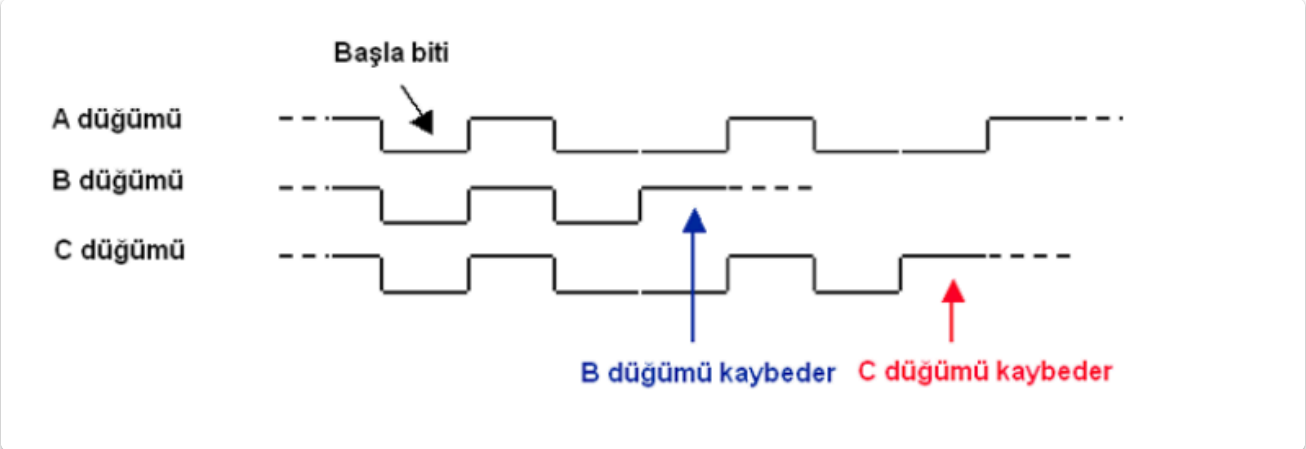
Mesaj ID temellidir.

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

Peki CAN-BUS'ta hatasız veri iletimi nasıl oluyor ?

- Herkes bir anda veri gönderirse çatışmalar olacağı açık.Bunun çözümü tüm nodelar daima iletim hattını dinler ve iletim hattının boş olduğu anı yakalamaya çalışır.
- CAN iletişim ortamına erişim yöntemi olarak bit öncelikli yapı ile CSMA/CD kullanır.
- Lojik 0'ın lojik 1 e baskın gelmesi sonucu küçük mesaj ID sine sahip mesajlar öncelik kazanırlar.
- Bir düğüm tarafından mesaj gönderilmesi kararlaştığında 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 işlemektedir.
- Eğer yol boşaldığında birden fazla düğüm yola mesaj yazmaya başlarsa düşük ID'li mesajı yazan düğüm yolu ele geçirir ve diğer düğümlerden aradan çekilerek tekrar göndermek üzere yolun boşalmasını beklerler.
- 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.
- Örneğin yola aynı anda veri yazmaya çalışan A,B ve C adında üç düğümümüz olsun. A düğümü yola 36 (100100), B düğümü 47 (101111) ve C düğümü 37 (100101) yazsın .



CANBUS MESAJLARI

- CAN sistemlerinde veriler paketlerin halinde iletlir. Ancak burada iki tip paketleme yapılır ve bu paketlemelerin özel adları vardır.
- 11 bit tanımlayıcıya sahip olanlar CAN2.0A diğer adıyla STANDART CAN, 29 bit tanımlayıcıya sahip olanlara ise CAN2.0B diğer adıyla EXTENDED (Geliştirilmiş) CAN denir. Aralarındaki temel fark ise tanımlanacak mesaj sayısıdır.

- Standart CAN'de 2^11 = 2048 mesaj tanımlanır.
- Extended CAN'de 2^29 = 536 870 912 mesaj tanımlanır.

BASE FRAME (Temel Çerçeve)

Her çerçeve SOF (Start of Frame) sinyali ile başlar.

- Bu sinyal 1 bitliktir ve dominattır.Ardından 12 bitlik denetim alanı gelmektedir.
- İlk 11 biti mesaj ID alanıdır ve bu alandaki ID değeri ile mesajlar etiketlenir.
- Denetim alanındaki son bir RTR diye adlandırılır ve özel anlamı vardır.
  - Bu bit 0 (dominant) ise gönderilen çerçeve veri çerçevesidir ve veri alanında, ID alanında tanımlanan mesaja ait veri vardır.
  - Bu bit 1 ise çerçeve istek çerçevesidir ve veri alanı yoktur.
- Bu çerçevenin ID alanındaki değer ile belirlenen mesaja ait veri ilgili düğümlerden istenmekt edir
- Bu çerçeveyi alan alıcı ID alanındaki değeri okuyarak hangi veriyi göndermesi gerektiğini anlar ve yol boşa çıktığında gönderir.Bu sayede CAN protokolü master ve slave olarak çalışabilmektedir.
- ID alanındaki ilk 7 bit ardışık olarak resesif(1) olamaz.

Kontrol Alanı

- Denetim alanından sonra kontrol alanı gelmektedir.
- Bu alanı ilk biti IDE diye isimlendirilir ve bu çerçevenin 11 bitlik ID alanına sahip 2.0A çerçevesi olduğunu belirten dominant bir bittir.
- Bu bitin ardından bir bitlik kullanılmayan rezerve alanı gelmektedir.
- Daha sonra 4 bitlik DLC diye isimlendirilen bir alan gelir.
  - DLC alanı gönderilen verinin kaç byte olduğunu söyler.

CRC Alanı

Veri alanını CRC alanı takip eder.

- Bu alan 16 bitliktir ve 15 bitlik CRC (Cyclic Redundancy Check) bilgisi ile resesif CRC Delimiter bitinden oluşmaktadır.
- CRC alanı gönderilen SOF alanına kadar gönderilen verinin doğru olup olmadığının anlaşılması için bir değerdir.
- Veriyi gönderen düğüm veri üzerinde bir takım işlemler yaparak 15 bitlik CRC değerini hesaplar ve çerçeveye ekler.

- Alan düğüm veriyi aldığında göndericinin yaptığı işlemler ile aynı işlemleri yapar ve CRC yi tekrar hesaplar. Alınan ve gönderilen CRC tutarlı ise veri doğru gönderilmiştir.
- Alıcı düğümlerden en az 1 tanesi bile veriyi yanlış aldıysa veri tekrar gönderilmelidir.

  ACK Alanı

  Bu alan iki bitliktir.
- İlk bitini gönderici resesif olarak gönderir.
- Eğer veri en az bir alıcı tarafından doğru alınmışsa alıcı yolda dominant biti yazar.
- Böylece gönderici mesajın en az bir alıcı tarafından alındığını anlar.
- Eğer gönderici dominant biti okuyamazsa ACK işaretinden kaynaklı bir hata olduğuna kanaat getirir ve veriyi tekrar yollar.
- Bu alanın ikinci biti ise ACK delimiter olarak adlandırılır ve resesiftir.

EOF Alanı

ACK alanının arkasından çerçevenin sonlandırıldığını belirten 7 bitlik EOF alanı gelir.

- Bu alandaki bitler resesiftir.
- Daha sonra ise çerçeveler arasında boşluk bırakmak amacıyla 3 bitlik INT alanı gelmektedir ve bitleri resesiftir.
- Böylece temel çerçevede bir mesaj gönderilmiş ve alınmış olur.

Extended Frame (Geliştirilmiş Çerçeve)

- Extended frame'de dominant SOF ile başlar.
- Ardından 2.0A da olduğu gibi 11 bitlik IDA alanı gelir.
- Ardından 2.0A'daki dominant RTR biti yerine resesif SRR biti gelmektedir.
- Ardından 2.0A'daki offsete denk gelecek şekilde IDE biti gelir.
  - Tek farkı 2.0B de bu bit resesiftir çünkü 29 bitlik ID ye sahip mesajlar iletilmektedir.
- IDE bitinden sonra 18 bitlik ikinci ID B alanı gelir.
- Ardından dominant RTR biti gelerek bu çerçevenin veri çerçevesi olduğunu belirtir.

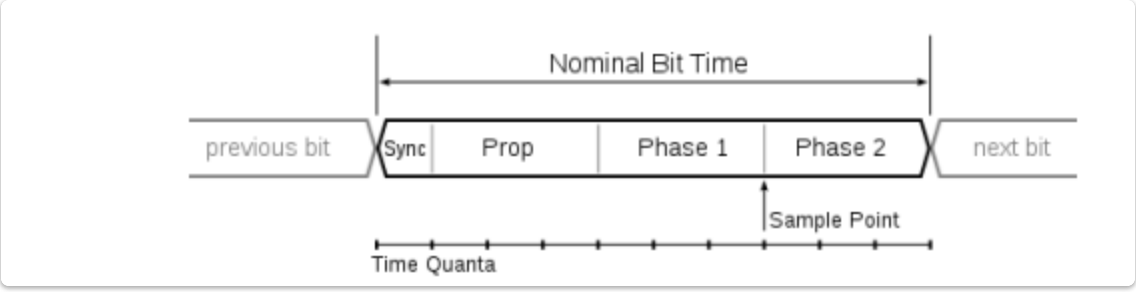
  Request Frame
- Çoğu zaman veri yolu okunan,oluşan bilgilerin gönderilmesi ile çalışmaktadır.
- Bazen düğümler istekte bulunurlar.
- Bunu istek çerçevesi ile yaparlar.
- İstek çerçevesi ile veri çerçevesinin 2 farkı vardır.
- Bunlar istek çerçevelerinde RTR biti resesiftir ve istek çerçevelerinin veri alanı yoktur.
- Kalan kısımlar veri çerçevesi ile aynıdır.

ERROR Frame

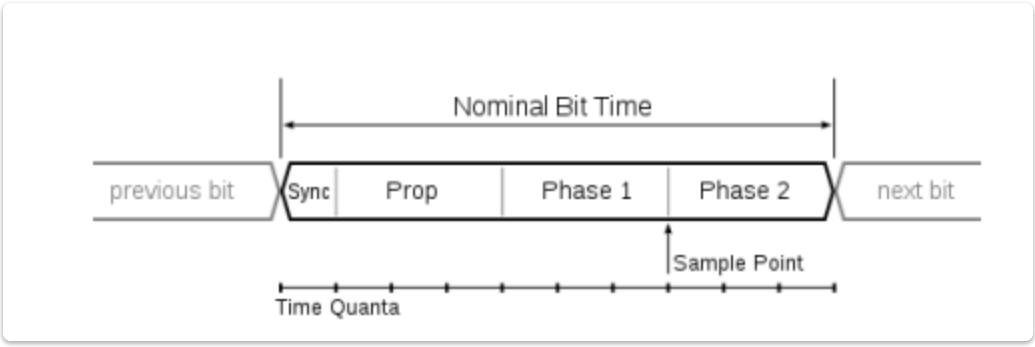
- Veri yolunda hata oluştuğunda hata çerçevesi gönderilmektedir.

- Hata çerçevesi iki alandan oluşmaktadır.
- Bu alanlar hata bayrakları (Error Flags) ve hata ayracı (Error Delimiter) alanıdır.
- Aktif ve pasif olmak üzere iki adet hata bayrağı vardır.
- Yolun durumuna göre aktif veya pasif hata oluşturmaktadır.
- CAN haberleşmesi haricinde clock yoktur.
- Senkronizasyon lojik değişimlere göre yapılmaktadır.
- Altı veya daha fazla adet aynı lojik seviyeden bitin ardışık okunması senkronizasyonun kayboluğu anlamına gelir.
- Bu yüzden aynı lojik seviyesine sahip her beş ardışık bit gönderildikten sonra araya karşı seviyeden bir bit sıkıştırılır.
- **Bunu CAN donanımı yapmaktadır ve bu işlemin adına bit stuffing denilmektedir.**

CAN BUS Bit Süresi



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



- Tseg1 ve Tseg2 oranı ayarlanarak örnekleme noktası bir bitlik zaman içerisinde kaydırılabilir.
- Bunu yapmamızdaki amaç iletim hattının uzunluğuna göre sistemin kararlı çalışabilmesini sağlamaktır. 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 kuanta süresi arasında değer alan bir değişken tarafından ayarlanır.
- Bit hızı aşağıdaki bağıntı ile hesaplanır(BRP=Baud Rate Prescaler).
- 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 ila 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=80 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 >= 5 TKUANTA ise SJW =4 tür.
- Tseg2 < 5 TKUANTA ise SJW = (Tseg2 1) TKUANTA 'dır.
- Bizim örneğimizde SJW=4 tür.

CAN BUS Filtre ve Maske

- CANBUS birimi, istenmeyen mesajları filtrelemek için kabul filtresi ve maske değerini kullanarak bu görevi yerine getirmek için ürün yazılımı içerir.

- Filtre maskesi , alınan çerçevenin tanımlayıcısındaki hangi bitleri karşılaştıracağını 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 taktirde çerçeve reddedilir.
- Örneğin yalnızca 00001234 kimliğini 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
- Örneğin 00001230 - 0000123F arasındaki ID ' li çerçeveleri almak istiyorsak maskeyi 1FFFFFF0 olarak ayarlamalıyız.
    - -> Filtre 00001230
    - -> Maske 1FFFFFF0
- 00001230 - 00001237 arasındaki çerçeveleri kabul etmek istiyorsak.
    - -> Filtre 00001230
    - -> Maske 1FFFFFF8
- **Tüm çerçeveleri kabul etmek istiyorsak**
    - -> Filtre 0x00000000
    - -> Maske 0x00000000
-