12 CAN

14 Ocak 2023 Cumartesi 17:55

12 CAN

Giriş

- Konu hakkında detaylı yazılmış yazılara http://www.barissamanci.net/Makale/15/can-bus-nedir-can-protokolu-incelemesi/, https://en.wikipedia.org/wiki/CAN_bus ve https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial linklerden ulaşabilirsiniz. Ayrıca https://youtu.be/SQgUDyu3dZw?si=pMJeZfnZgAG9leOS linkteki videoyu da izleyebilirsiniz.

Nedir

- Açılımı **Control Area Network Bus** yani Kontrol Alan Ağı Veri Yolu'dur.
- 1980'li yıllarda Robert Bosch tarafından **otomotiv** de kablo yumağı yerine bir kablodan yazılım kontrollü veri transferini sağlamak amacıyla geliştirilmiştir.
- 1987 ortaları itibari ile de Intel ve Philips gibi yarıiletken üreticileri CAN çiplerini piyasaya sunmaya başladılar. Böylelikle başta otomobiller olmak üzere her türlü taşıtlardan sanayi ürünlerine kadar birçok alanda CAN en yaygın kullanılan **veri yolu protokolü** haline geldi.
- **Güvenliğin** çok önemli olduğu **gerçek zamanlı** uygulamalarda kullanılır. Öyle ki istatistiksel olasılık hesapları sonucunda bir asırda bir tane tespit edilemeyen mesaj hatası yapabileceği hesaplanmıştır.
- Uygulama alanı yüksek hızlı ağlardan düşük maliyetli çoklu kablolamalı sistemlere kadar geniştir.
- CAN BUS 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 **40 m'de 1Mbit/sn** iken 1 km 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.

Katmanlar

- CAN BUS sistemleri genellikle **Nesne Katmanı** (Object Layer) , **İletim Katmanı** (Transfer Layer), **Fiziksel Katman** (Physical Layer) olmak üzere üç ana katmana ayrılır
- 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.
    Örneğin; frame kontrolü, mesaj önceliği belirleme, hata kontrolü, hata sinyalleşmesi ve hata kapatma.
  - İletim katmanı yeni bir mesajı yollamadan önce iletim hattının boş olmasına dikkat eder. Aynı zamanda iletim hattından veri alınmasından da sorumludur.
  - Ayrıca senkron iletişim için veri transferi sırasında bit zamanlamasının bazı parametrelerini göz önünde bulundurur.
  - CAN BUS üzerinden haberleşen tüm sistem bileşenlerine ünite (nod) denir.
- Fiziksel Katmanın görevi,
  - Fiziksel katman, üniteler arasında veri haberleşmesi sırasındaki tüm elektriksel kısımdır.

Genel Özellikler

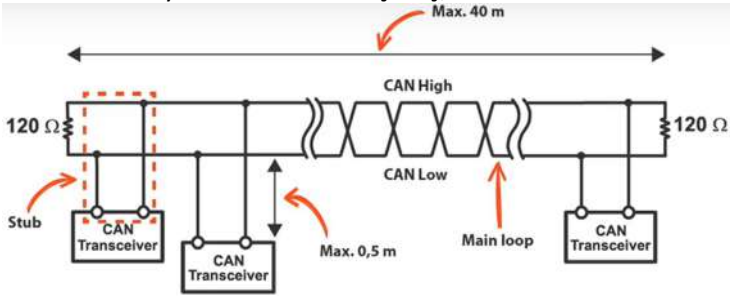
- Mesaj Önceliği
- Kayıp Zaman Güvenliği
- Yapılandırma Esnekliği
- Senkronizasyonlu çoklu kabul: Aynı veri birçok ünite tarafından alınabilir.
- Sistemdeki veri yoğunluğunu kaldırabilme
- Çok efendili (Multi master) çalışma
- Hata tespiti ve hataya ilişkin sinyalleri üretme
- Mesaj yollanmasında hata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

Çalışma Mantığı

- **Haberleşme Yapısı,**
  - Standart olarak **half-duplex** haberleşme yapısını kullanır.
  - Aynı anda yalnızca bir cihaz veri gönderebilir; diğer cihazlar bu sırada gönderim yapamaz, yalnızca dinleyebilir.
- **Çakışma Önleme Mekanizması,**
  - Çakışmaları önlemek için **CSMA/CR (Carrier Sense Multiple Access with Collision Resolution)** mekanizmasını kullanır:
    - Bir cihaz veri göndermek istediğinde, önce iletişim hattının boş (IDLE) olup olmadığını kontrol eder.
    - Hattın boş olduğunu algılarsa veri göndermeye başlar.
    - Eğer başka bir cihaz aynı anda veri göndermeye başlarsa, çakışma çözümü için **çerçeve önceliği** devreye girer.
      - CAN protokolünde daha düşük ID'ye sahip olan mesajın önceliği daha yüksektir ve bu mesaj gönderilmeye devam eder.
- **Çatışma Durumu ve Çözüm,**
  - Birden fazla cihaz aynı anda veri göndermeye çalışırsa çakışma oluşabilir.
  - Çakışma durumunda:
    - Çakışmaya dahil olan cihazlar, iletim ortamını dinleyerek çerçeve önceliğine göre bir çözüm uygular.
    - Düşük öncelikli mesaj gönderen cihazlar hattı serbest bırakır ve hattın boşalmasını bekler.
    - Yüksek öncelikli mesaj gönderilmeye devam eder.
- **Mesaj Önceliklendirme,**
  - Mesaj öncelikli bir sistemdir.
    - İnternet protokollerinde cihazlara numara atanırken, CAN protokolünde mesajlara numara atanır.
    - Bu numaralar, mesajların öncelik sırasını belirler.
  - Daha düşük ID'li mesajlar, daha yüksek önceliğe sahiptir ve iletim hattına önce gönderilir.
- **Multi-Master Tasarımı,**
  - CAN-BUS, **multi-master** yapıya sahiptir.
    - Tüm cihazlar, eşit öncelikli olarak veri gönderme hakkına sahiptir.
    - Ancak önceliklendirme mekanizması sayesinde çakışmalar çözülür.
- **Veri Alımı ve Filtreleme,**
  - CAN-BUS hattına gönderilen tüm veriler, tüm cihazlar tarafından alınır.
    - Her cihaz, içerisindeki filtreleme mekanizması ile yalnızca kendisini ilgilendiren mesajları işler.
    - İlgisiz mesajlar, filtreleme sistemi tarafından göz ardı edilir.
- **Hız ve Mesafe Sınırları,**
  - Veri iletim hızı ve hattın uzunluğu arasında bir ilişki vardır:
    - **1 Mbit/sn** hızında iletişim için maksimum hat uzunluğu **40 metre** ile sınırlıdır.
    - **40 Kbit/sn** hızında iletişim için maksimum hat uzunluğu **1 kilometreye** kadar çıkabilir.

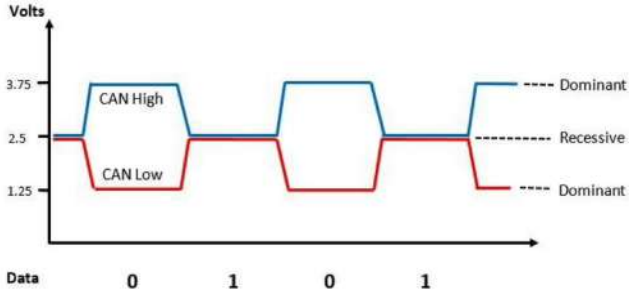
Veri Yolu ve Bağlantı Noktaları

- Genel olarak bir CAN bağlantı noktası aşağıda gösterilmiştir. Bağlantı noktaları **düğüm** olarak isimlendirilirler. Mikrodenetleyici ve CAN kontrolcüsünden oluşmaktadırlar.
- Bizim sistemlerimizde bu CAN kontrolcüsü çipe dahil edilmiştir fakat harici entegre olarakta kullanılabilir.
- CAN kontrolcüsü CAN veri yoluna direk bağlanır.
- Bu veri yolu **iki telden** oluşmuş, iki tarafı **120 Ohm** dirençlerle sonlandırılmıştır.



- CAN yapısında alıcı ve verici birbirinden fiziksel olarak **bağımsızdır**. Fakat düğümlerin yapısı gereği gönderilen mesaj alıcıdan dinlenebilmektedir. Bu sayede veriyi gönderen işlemci gönderdiği veri ile okuduğu

veriyi karşılaştırarak **hata** ve mesajların **öncelik** seviyelerine göre iletilmelerine olanak sağlanır. Multi master yapıda çalışabilen bu sistem için bu özellik önemlidir.

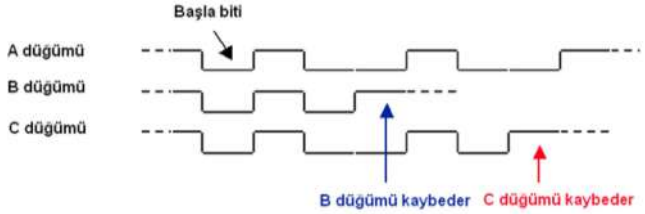


- CAN IDLE modda ise yani veri iletişimin aktif olmadığı durumda CANH ve CANL hatları 2.5V seviyesindedir. Data transfer edilmeye başladığı anda CANH hattı 3.75V seviyesine yükselirken, CANL ise 1.25V seviyelerine kadar düşmektedir. Bu durumda her iki sinyal seviyesi arasında 2.5V görülmektedir. Buradaki iletilişim iki veri yolu hattındaki voltaj farkına dayanır.

	dominant	**recessive**
dominant	dominant	dominant
recessive	dominant	recessive

	0	**1**
0	0	0
1	0	1

- Yukarıdaki şekilde görüldüğü gibi hattın lojik seviyesi **iki farklı değer** alabilmektedir ve bu değerlerin seviyesi gözükmektedir.
- Lojik 1, recessive (çekingen); lojik 0 dominant (baskın) olarak adlandırılmaktadır.
  Bunun sebebi hatta farklı düğümlerden aynı anda 0 ve 1 yazılması durumunda **0'ın 1 karşı baskın** gelmesidir.
- Lojik 0'ın lojik 1'e 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 boşalmasını 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.

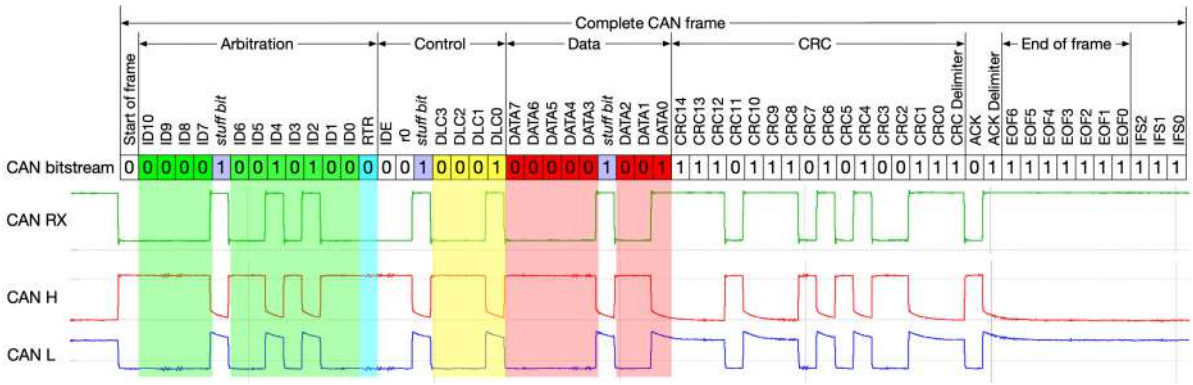


  - A düğümü mesajını gönderdikten sonra daha önemli bir mesaj yoksa C düğümü sonrada B düğümü mesajını gönderir.

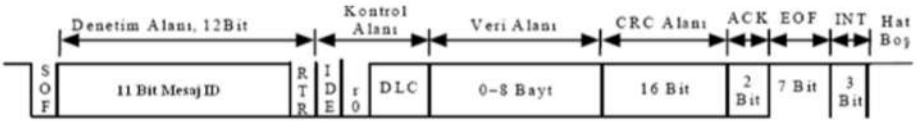
Mesajlar

- CAN sistemlerinde veriler paketlerin halinde iletilir. 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** 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.
- Bu bilgilerin tutulduğu alana **Mesaj ID** alanı denir. **Mesaj önceliğini tanımlamada** buradaki sayı dikkate alınır.
- Veri yolunda mesajlar çerçevelere bölünerek iletilmektedir. İki farklı mesaj türü vardır.
  - Bunlar **Veri Çerçevesi (Message Frame)** ve **İstek Çerçevesi (Remote Transmit Request Frame)**'dir.
  - Farkları ise veri çerçevelerinin en fazla 8 byte uzunluğa kadar veri taşıyabilmesi, istek çerçevesinde ise belli bir mesaja ait verinin istenmesidir.

Frame



Base Frame



Denetim Alanı

- Her çerçeve **SOF (Start Of Frame)** sinyali ile başlar. Bu sinyal 1 bitlik dominanttır.
- Ardından 12 bitlik **Denetim Alanı** gelmektedir.
    - 11 bit **Mesaj ID** alanıdır ve bu alandaki ID değeri ile mesajlar etiketlenir. ID alanındaki ilk 7 bit ardışıl olamaz.
    - Denetim alanındaki son bit **RTR** diye adlandırılır ve özel anlamı vardır. Bu bit dominant ise gönderilen çerçeve **veri çerçevesi**dir ve veri alanında, ID alanında tanımlanan mesaja ait veri vardır demektir. Eğer bu bit 1 ise çerçeve, **İstek Çerçevesi'**dir ve veri alanı yoktur.
- Bu çerçevenin ID alanındaki değer ile belirlenen mesaja ait veri ilgili bilgi, düğümlerden istenmektedir.
- Bu çerçeveyi alan, ID alanındaki değeri okuyarak hangi veriyi göndermesi gerektiğini anlar ve yol boşa çıktığında gönderir. Bu sayede CAN protokolü master ve slave olarak çalışabilmektedir.

Kontrol Alanı

- Denetim alanından sonra **Kontrol Alanı** gelmektedir.
- Bu alanın ilk biti **IDE** diye isimlendirilir ve bu çerçevenin 2.0A çerçevesi olduğunu belirten dominant bir bittir.
- Bu bittin ardından bir bitlik kullanılmayan **rezerve alan** gelmektedir.
- Daha sonra 4 bitlik **DLC** diye isimlendirilen bir alan gelir. DLC alanı gönderilen verinin **kaç byte** olduğunu söyler.

Veri Alanı

- Kontrol alanını veri alanı takip etmektedir.
- Veri alanı en fazla 8 byte olabilmektedir.

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 alanından CRC alanına kadar gönderilen verinin doğru olup olmadığının anlaşılması için bir değerdir.
- Veriyi gönderen düğüm veri üzerinde bir takip işlemler yaparak 15 bitlik CRC değerini hesaplar ve çerçeveye ekler.
- Alan düğüm veriyi aldığında göndericinin yaptığı işlemleri 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

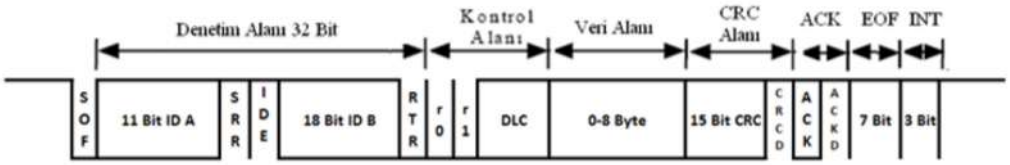
- CRC alanını ACK alanı takip eder. Bu alan 2 bitliktir.
- İlk bitini gönderici resesif olarak gönderir. Eğer veri en az bir alıcı tarafından doğru alınmışsa alıcı yola dominant bit yazar. Böylece gönderici mesajın en az bir alıcı tarafından alındığını anlar. Eğer gönderici dominant biti okuyamasa ACK işaretinden kaynaklı bir hata olduğuna kanaat getirir ve veriyi tekrar yollar.
- Bu alanın ikinci biti ise ACK delimeter olarak adlandırılır ve resesiftir.
  Bunun bir mesaj iletimiyle açıklamalı. Gönderici başla biti ile iletim hattında şuan gönderici benim der. Ardından Mesaj ID Alanı, Veri Alanı, CRC Alanı gönderilir. Alındı Bilgisi Alanında ise iletim ortamı

resesif(çekinik) tutulur. Eğer tüm ünitelerden biri, mesaj onu ilgilendirse ya da ilgilendirmese dahi, mesajı alabiliyorsa iletim ortamını dominant(baskın) yapar ve böylece gönderici en az bir ünite veriyi alabildiği için bitir bitini yollayıp iletim ortamını diğer ünitelerin kullanımına bırakır. Yani Alındı Bilgisi Alanında "Aldınız mı?" sorusuna yanıt beklenir. Eğer Alındı Bilgisi sürecinde herhangi bir üniteden alındığına dair bilgi alınmazsa ACK hatasından kaynaklı hata oluştuğunu belirten Hata Çerçevesi üretilir ve gönderici tekrar yollanmaya çalışır.
Eğer gönderen İstek Çerçevesi yollamışsa, alıcı da iletim hattının boş bir anında cevabını göndericiye yollar.

EOF Alanı

- ACK Alanının arkasından çerçevenin sonlandırıldığını belirten 7 bitlik **EOF** (End of Frame) alanı gelir.
  Bu alandan 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



- CAN2.0B'de mesaj ID si 29 bittir.
- Geriye uyumluluk açısından CAN2.0B geliştirilirken 2.0A göz önünde bulundurularak geliştirilmiştir.
- İki protokolde aynı yolda çalışabilmektedir fakat 2.0A düğümlerine 29 bitlik ID'li mesajlar gönderilmemelidir.

Denetim Alanı

- 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 ofsete 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.

Kontrol Alanı

- Denetim alanından sonra kontrol alanı gelmektedir.
- Kontrol alanının ilk iki biti **rezerv alandır** ve kullanılmaz.
- Son dört bit **DLC** alanını oluşturur ve gönderilen verinin kaç byte olduğunu söyler.

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.

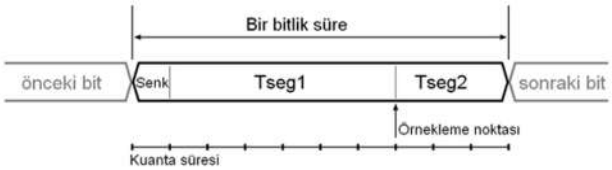
Error Frame

- Veri yolunda hata oluştuğunda hata çerçevesi gönderilmektedir.
- Hata çerçevesi iki alandan oluşmaktadır.
- Bu alanlar hata bayrakları (Error Flags) ve hata ayıracı (Error Delimiter) alanıdır.
- Aktif ve pasif olmak üzere iki adet hata bayrağı vardır.
- Yolun durumuna göre aktif veya pasif hata oluşturulmaktadır.
- CAN haberleşmesinde harici clock yoktur.
- Senkronizasyon lojik değişimlere göre yapılmaktadır.
- Altı veya daha fazla adet aynı lojik seviyeden bitin ardışıl okunması, senkronizasyonun kaybolduğu anlamına gelip alıcıda hata oluşturmaktadır.
- Bu yüzden aynı lojik seviyede beş ardışıl bit gönderildikten sonra araya karşı seviyeden bir bit sıkıştırılır.
- Bunu CAN donanımı yapmaktadır ve bu işlemin adına bit stuffing denilmektedir.

Bit Time

- Çoğu diğer seri protokolün aksine, CAN protokolünde bit hızı direkt 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 1 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 iletişim hattının uzunluğuna göre sistemin kararlı çalışabilmesini sağlamaktır.
    - Uzun iletim hatları kullanıyorsak örnekleme noktası geri çekilmelidir.
        - Örnekleme noktası neden ileri alınır?
            - Osilatör hassasiyeti düşük ise örnekleme noktası ileri kaydırılır.
            - Bus üzerindeki bit hızındaki ufak sapmaları telafi eder.
            - Ek olarak alıcı bit zamanlamalarını ayarlayarak vericiye kilitlenebilirler.
- Her bit, kullanıcı tarafından ayarlanabilen **synchronous jump width** denilen 1 ile 4 kuanta süresi arasında değer alan bir değişken tarafından ayarlanabilir.
- Bit hızı aşağıdaki bağıntı ile hesaplanır.

  $$\text{Baud Rate} = P_{CLK} / (\text{Prescaler} * (1 + Tseg1 + Tseg2))$$
    - Örneğin, Baud Rate 125kHz (125000 bit/s), PCLK'yi 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.

      $$\text{Prescaler} = PCLK / (\text{Bit Rate} * KUANTA)$$
    - Bilinen değerlerimizi denkleme yerine koyalım.

      $$\text{Prescaler} = 60M / (125k * KUANTA)$$
    - Bir bitlik periyodun 8 ila 25 kuanta arasında olduğunu biliyoruz.
      Bu bilgiyi kullanarak Prescaler tam sayı olacak şekilde KUANTA yerine 8 ile 25 arasında uygun bir sayı seçelim.
      KUANTA = 16, Prescaler = 30 olacak şekilde denklemi çözeriz.
    - Şimdi Tseg1 ile Tseg2 arasındaki oranı ayarlayalım.

      $$16 = (1 + Tseg1 + Tseg2)$$
    - Örnekleme noktası oranı periyodun %70'ine denk gelmesi için,

      $$\text{Örnekleme Noktası} = (KUANTA * 70) / 100$$

      Dolayısıyla 16 * 0.7 = 11.2 olur. Buradan Tseg1 = 10 için ve Tseg2 = 5 olarak bulunur. Bu durumda örnekleme noktası %68.8'e denk gelir.
    - Synchronous jump width değeri de aşağıdaki şekilde hesaplanır.
      Tseg2 >= 5 TKUANTA ise SJW = 4 tür.
      Tseg2 < 5 TKUANTA ise SJW = (Tseg2 -1) TKUANTA'dır. Bizim örneğimizde SJW = 4'tür.

Hata Önleme

- CAN protokolünün **beş adet** hata önleme yöntemi vardır.
- Herhangi bir hata oluştuğunda gönderici veriyi tekrar gönderir böylece işlemcinin olaya müdahale etmesine gerek kalmaz.
- Hata önleme yöntemlerinden 3 tanesi **çerçeve** düzeyinde, 2 tanesi ise **bit** düzeyindedir.
- Çerçeve düzeyinde olanlar **çerçeve formatı**, **CRC** ve **ACK** kontrolüdür.
- Bit düzeyinde olanlar ise **bit** ve **bit stuffing** kontrolüdür.

Frame Kontrolü

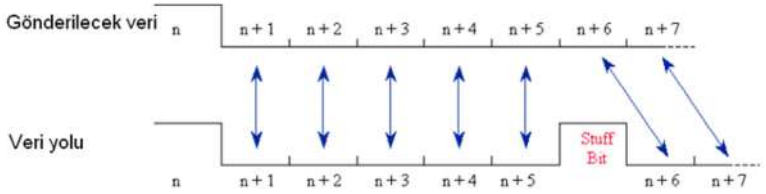
- Çerçeve formatı kontrolünde, alıcı veriyi aldıktan sonra verinin formatını kontrol eder ve çerçeve yapısı ile uyumlu olup olmadığını karşılaştırır. Alınan veride eksik alan varsa veri reddedilir ve veri yoluna hata çerçevesi bırakılır. Bu sistem doğru formatta veri alımını sağlar.
- CRC kontrolünde, SOF bitinden CRC bitlerinin başına kadar olan bitler bir takım işlemlerden geçirilerek CRC

kodu üretilir. Alıcıda bu CRC kodu alınan veri ile karşılaştırılır ve alınan bitlerin doğru olup olmadığını sınar. Format kontrolünden sonra alıcının bitleri kontrol etmesi ile formata uyan fakat hatalı mesajların önüne geçilmiş olur.

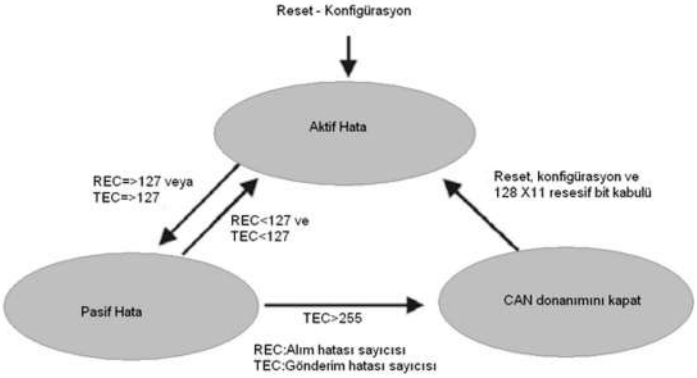
- ACK kontrolünde ACK mesajının göndericiye ulaşmamasıdır. ACK bitinin anlamı alınan mesajı alıcının onaylamasıdır. Gönderici CRC bitlerini gönderdikten sonra ACK bitini resesif olarak gönderir. Alıcılardan en az bir tanesinin hattaki resesif olan ACK bitini dominant bitle ezmesi beklenir. Eğer zaman aşımı sonucu ACK biti göndericiye ulaşmamışsa ACK bitinde hata olduğu şeklinde yorumlanır. Bunun sonucu göndericide hata oluşur ve ACK onayını alana kadar aynı mesajı tekrar gönderir.

Bit Kontrolü

- Bit stuffing hatasında, gönderici ile alıcı arasında saat darbeleri gönderilmez. Bunun yerine veri yolundaki CANL ve CANH hatlarındaki lojik değişimler ile senkronizasyon sağlanır. Bunun sonucu olarak aynı lojik seviyeden 5'ten fazla bitin art arda gelmesi senkronizasyonun bozulduğu anlamına gelir ve alıcıda hataya sebebiyet verir. Bunu engellemek için gönderici aynı 5 seviyeden sonra karşı seviyeden bir bit göndererek iletişime devam eder.
- Bunun sonucu olarak herhangi bir düğüm herhangi bir anda hata mesajı oluşturmak istediğinde veri yoluna 6 adet dominant bit yazar ve hataya sebebiyet verir.



- Bit düzeyindeki diğer hata ise bit kontrolüdür. Veri yolu boşaldığında düğümlerin mesaj göndermek için veri yolunu mesajların ID'lerine göre ele geçirdiğini söylemiştik. Her düğüm veri yoluna yazdığı biti tekrar geri okuyarak kendisinin gönderdiğinden daha önemli bir mesaj var mı diye bakar. Eğer daha önemli mesaj varsa geri çekilerek veri yolunun boşalmasını bekler. Böylece veri yolunu bir düğüm kazanmış olur ve bitleri göndermeye devam eder. Aynı zamanda da gönderdiği bitleri geri okur. Eğer veri yolunu kazanan düğüm gönderdiği seviyeden farklı bir seviye okursa hata oluşturur.
- CAN yapısının temelinde her mesajın farklı bir ID ile etiketlendiğini söylemiştik. Mesajları etiketlerken bu kurala dikkatle uymamız gerekmektedir. Yoksa farklı düğümlerin aynı ID ye ait farklı veriler göndermesi alıcıda CRC hatasına sebep olacaktır.



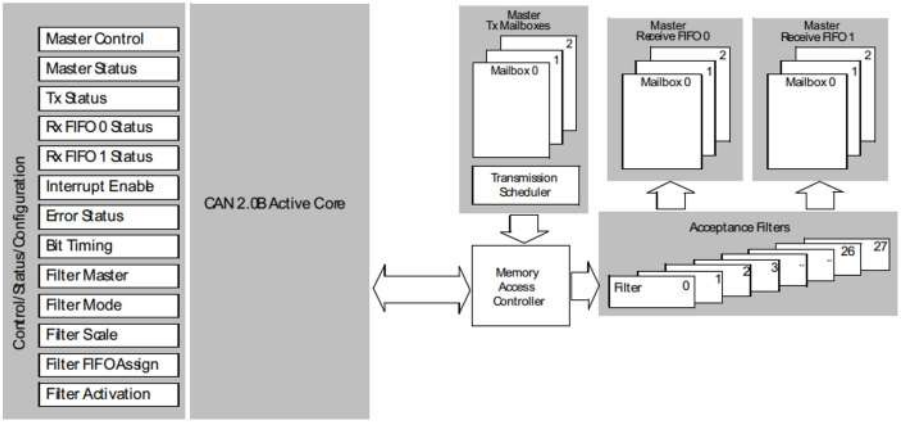
- CAN donanımı oluşan hatalara göre hata durumları arasında geçiş yapmaktadır. **İki adet hata sayıcısı** vardır.
- Bunlar göndericide oluşturulan hataları ve alınan hataları sayarlar.
- Herhangi bir sayıcı 127 ve büyük bir değere ulaşırsa donanım pasif hata moduna girer.
- Bu modda gelen hata çerçevelerini cevaplamaya devam eder fakat hata oluşturduğunda dominant bitler yerine resesif bitler gönderir.
- Eğer gönderim hata sayısı 255'i geçerse donanım kapatılır ve hattaki iletişime karışmaz ve etkilenmez.
- Haberleşmeyi tekrar başlatmak için işlemcinin olaya müdehale etmesi gerekmektedir.
- CAN donanımını resetler ve tekrar konfigüre eder. Bu mekanizmalar düğüm hatalı duruma düştüğünde art arda hata mesajları göndererek veri yolunu meşgul etmesini önlemek içindir.

Birim Yapısı

- STM32 serisi mikrodenetleyiciler gelen mesajları donanım seviyesinde işler ve yalnızca filtreyi geçen mesajlar posta kutusuna iletilir. Bu, işlemcideki yükü azaltmak ve ekstra kesme işlemine dikkat etmememizi sağlar.
- **bxCAN** adlı **Basic Extended CAN**, bir CAN ağ arabirimidir. CAN 2.0A ve 2.0B sürümlerini destekler. Çok sayıda gelen mesajı maksimum verimlilikle ve en az işlemci kullanımıyla yönetmek için tasarlanmıştır. Ayrıca,

mesajların transferi için öncelikli şartlardan da sorumludur.

- bxCAN modülü, CAN mesajlarının iletimini ve alımını tamamen otonom bir şekilde gerçekleştirir. Standart tanımlayıcılar (11 bit) ve genişletilmiş tanımlayıcılar (29 bit) donanım tarafından tamamen desteklenir.
- bxCAN'ın çalışma yöntemi aşağıda verilmiştir.

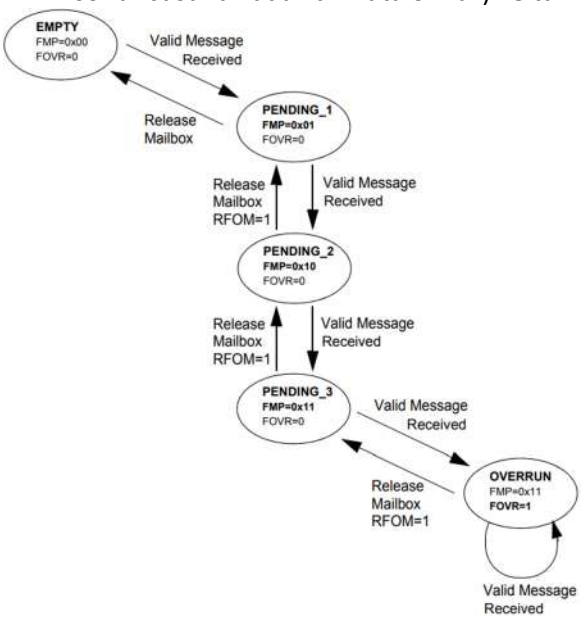


Mesaj Alımı

- 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.

Geçerli Mesaj

- Alınan bir mesaj, CAN protokolüne göre doğru bir şekilde alındığında geçerli sayılır (EOF alanının sonuncusuna kadar bir hata olmaz) ve tanımlayıcı filtrelemeden başarıyla geçer.



FIFO Yönetimi

- 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 doluysa), bir sonraki geçerli mesaj alımı aşılmaya neden olacak ve bir mesaj kaybolacaktır.
- Donanım, CAN_RFR kaydında FOVR bitin iayarlayarak 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.

Alımla İlgili Kesmeler

- 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 üreteceğine 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.

Filtreleme

- Mesajların CAN veri yolundan nasıl işlendiğini öğrendik. Mesaj alma ve posta kutusuna gönderme koşullarından biri mesaj tanımlayıcısını bir filtreden geçirmektir.
- CAN protokolünde bir mesajın tanımlayıcısı bir düğümün adresi ile ilişkili değil, mesajın içeriği ile ilişkilidir.
- Sonuç olarak bir verici mesajını tüm alıcılara yayınlar.
- Mesaj alımında bir alıcı düğüm tanımlayıcı değerine bağlı olarak yazılımın mesaja ihtiyaç duyup duymadığına karar verir. Mesaj gerekiyorsa, SRAM'ye kopyalanır. Aksi takdirde, mesaj yazılıma müdahale edilmeden atılmalıdır.
- Bu gerekliliği yerine getirmek için, bxCAN Denetleyicisi, uygulamaya 28 yapılandırılabilir ve ölçeklenebilir filtre bankası sağlar.
- Bu donanım filtreleme, yazılım tarafından filtreleme yapmak için gerekli olacak CPU kaynaklarını korur.
- Her filtre bankası, iki adet 32-bit CAN_FxR0 ve CAN_FxR1 yazmaçlarından oluşur.
- Filtreleri uygulamaların ihtiyaçlarına göre optimize etmek ve uyarlamak için, her filtre bankası birbirinden bağımsız olarak ölçeklendirilebilir.

Ölçeklenebilir Genişlik

- Ölçek filtresine bağlı olarak, filtre bankası şunları sağlar:
    - STID[10:0], EXID[17:0], IDE ve RTR bitleri için bir 32 bit filtre
    - STID[10:0], RTR, IDE ve EXID[17:15] bitleri için iki adet 16 bit filtre.
- IDE biti (Tanıtıcı Genişletme Bit), filtrenin uzatılmış mesaj çerçevesi için tasarlandığını ve Uzaktan İletim İsteği (RTR) bitinin "Uzak" ileti türünü gösterdiğini anlamına gelir.
- Ek olarak, filtre **maske** modunda veya **listeleme** modunda yapılandırılabilir.

Maske Modu

- Maske modunda, tanımlayıcı kayıtları, tanımlayıcının hangi bitlerinin "eşleşmesi gerektiği" veya "umurumda değil" olarak ele alındığını belirten maske kayıtları ile ilişkilendirilir.
- Gelen mesajın kimliği (ID) ve maske değerleri (Mask ID) karşılaştırılır. Maske, hangi bitlerin kontrol edileceğini belirler.
    - Maske: 0xFFFF → Tüm bitler kontrol edilir.

Identifier Liste Modu

- Tanımlayıcı listesi modunda, maske kayıtları tanımlayıcı kayıtları olarak kullanılır.
- Böylece bir tanımlayıcı ve maske tanımlamak yerine, tek tanımlayıcı sayısını iki katına çıkaran iki tanımlayıcı belirtilir.
- Gelen tanıtıcının tüm bitleri, filtre kayıt defterinde belirtilen bitlerle eşleşmelidir.
- Gelen mesaj, filtre bankasında tanımlanan sabit bir listeyle karşılaştırılır. Sadece eşleşen mesajlar kabul edilir.
    - Liste: 0x123, 0x456 → Sadece bu iki mesaj kabul edilir.

Örnek Çalışma Modları

- CAN 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 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 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.
- Örneğin 00001230 - 0000123F arasındaki ID 'li çerçeveleri almak istiyorsak maskeyi 1FFFFFF0 olarak ayarlamalıyız.

Filtre	0x00001230
Maske	0x1FFFFFFO

- Yalnızca 00001230 - 00001237 arasındaki çerçeveleri kabul etmek istiyorsak maskeyi 1FFFFFF8 olarak ayarlamalıyız.

Filtre	0x00001230
Maske	0x1FFFFFF8

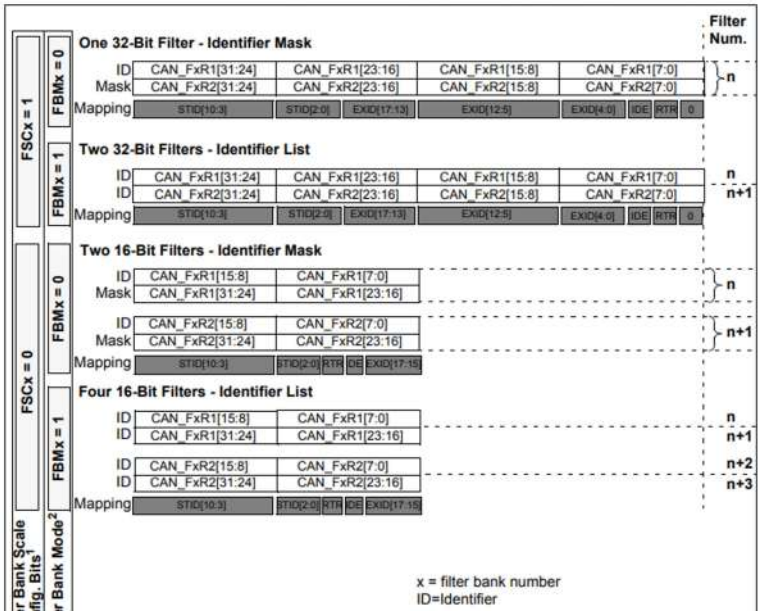
    - Bir çerçeve geldiğinde ID'si filtre ile karşılaştırılır ve 0.bit haricindeki tüm bitlerin eşleşmesi gerekir ve ayrıca 0.bit de 0 ile 8 arasında olmalıdır.
- Örneğin herhangi bir gelen çerçevenin kabul edilmesi isteniyorsa filtreyi ve maskeyi sıfıra ayarlamalıyız.

Filtre	0x0000/0
Maske	0x0000/0

    - İşlemin sonucunda tüm çerçeveler kabul edilir.

Filtre Bankası Ölçeği ve Mod Yapılandırması

- Filtre sıraları, karşılık gelen CAN_FMR kaydı vasıtasıyla konfigüre edilir.
- Bir filtre bankasını konfigüre etmek için, CAN_FAR kaydındaki FACT bitini temizleyerek etkisizleştirilmelidir.
- Filtre ölçeği, CAN_FS1R kaydındaki karşılık gelen FSCX biti vasıtasıyla yapılandırılmıştır.
- İlgili Maske/Tanımlayıcı kayıtları için tanımlayıcı listesi veya tanımlayıcı maske modu, CAN_FMR kaydındaki FBMx bitler vasıtasıyla yapılandırılmaktadır.
- Bir tanımlayıcı grubunu filtrelemek için, Maske/Tanımlayıcı kayıtlarını maske kipinde yapılandırın.
- Tek tanımlayıcıları seçmek için, Maske / Tanımlayıcı kayıtlarını tanımlayıcı liste modunda yapılandırın.
- Uygulama tarafından kullanılmayan filtreler devre dışı bırakılmalıdır.
- Bir filtre bankası içindeki her bir filtre, moda ve filtre sıralarının her birinin ölçeğine bağlı olarak O'dan maksimuma kadar numaralandırılır (Filtre Numarası olarak adlandırılır)



Filter Bank Scale Config. Bits[1]	Filter Bank Mode[1]

x = filter bank number
ID=Identifier
[1] These bits are located in the CAN_FS1R register
[2] These bits are located in the CAN_FM1R register

Filtre Index Eşleştirme

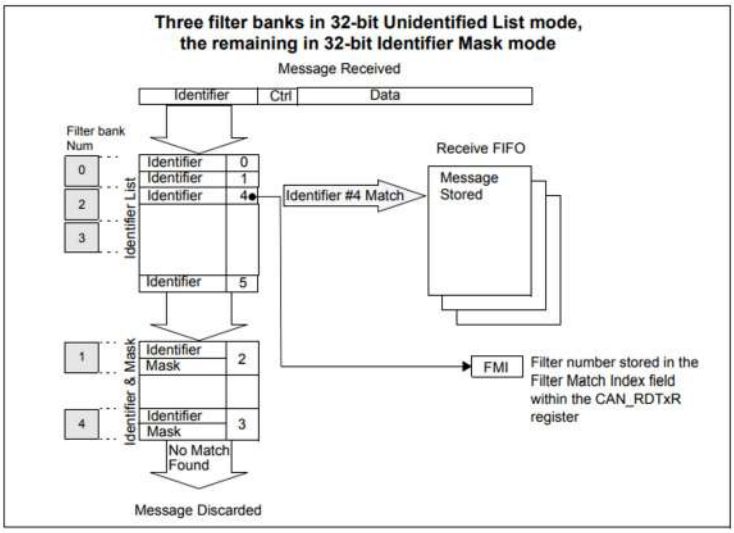
- FIFO'ya bir mesaj alındıktan sonra uygulamaya geçilebilir. Genellikle, uygulama verileri SRAM konumlarına kopyalanır.
- Verileri doğru yere kopyalamak için, uygulamanın verileri tanımlayıcı aracılığıyla tanımlaması gerekir. Bundan kaçınmak ve SRAM konumlarına erişimi kolaylaştırmak için CAN denetleyicisi bir Filtre Eşleme Dizini sağlar.
  • Bu indeks, filtre öncelik kurallarına göre mesajla birlikte posta kutusunda saklanır. Böylece, alınan her mesajın, ilişkili filtre eşleşme endeksi vardır.
- Filtre Eşleştirme dizini iki şekilde kullanılabilir:
  - Filtre Eşleme dizinini beklenen değerlerin bir listesiyle karşılaştırın.
  - Veri hedefi konumuna erişmek için bir dizideki dizin olarak Filtre Eşleştirme Dizinini kullanın.
- Maskelenmemiş filtreler için, yazılımın artık tanımlayıcıyı karşılaştırması gerekmez. Filtre maskelenmişse, yazılım yalnızca maskeli bitlerle karşılaştırmayı azaltır.
- Filtre numarasının endeks değeri, filtre sıralarının aktivasyon durumunu dikkate almaz.
- Ek olarak, her FIFO için bir tane olmak üzere iki bağımsız numaralandırma şeması kullanılır.

Filter Bank	FIFO0	Filter Num.
0	ID List (32-bit)	0, 1
1	ID Mask (32-bit)	2
3	ID List (16-bit)	3, 4, 5, 6
5	Deactivated ID List (32-bit)	7, 8
6	ID Mask (16-bit)	9, 10
9	ID List (32-bit)	11, 12
13	ID Mask (32-bit)	13

Filter Bank	FIFO1	Filter Num.
2	ID Mask (16-bit)	0, 1
4	ID List (32-bit)	2, 3
7	Deactivated ID Mask (16-bit)	4, 5
8	ID Mask (16-bit)	6, 7
10	Deactivated ID List (16-bit)	8, 9, 10, 11
11	ID List (32-bit)	12, 13
12	ID Mask (32-bit)	14

Filtre Öncelik Kuralları

- Filtre kombinasyonuna bağlı olarak bir tanımlayıcının birkaç filtreden başarılı bir şekilde geçtiği ortaya çıkabilir. Bu durumda, alıcı posta kutusunda depolanan filtre eşleşme değeri aşağıdaki öncelik kurallarına göre seçilir:
  - 32 bit filtre, 16 bit filtreye göre önceliklidir.
  - Eşit ölçekte filtreler için, Tanımlayıcı Maskesi moduna göre Tanımlayıcı Listesi moduna öncelik verilir.
  - Eşit ölçek ve moddaki filtreler için öncelik, filtre numarasından verilir (sayı ne kadar küçükse, öncelik o kadar yüksek olur).



- Bir mesajın alınmasında, tanımlayıcı ilk önce tanımlayıcı liste modunda konfigüre edilen filtrelerle karşılaştırılır.
- Bir eşleşme varsa, mesaj ilişkili FIFO'da saklanır ve eşleşen filtrenin indeksi Filtre Eşleştirme Dizininde

saklanır.

- Örnekte gösterildiği gibi, tanımlayıcı Tanımlayıcı # 2 ile eşleşir, böylece mesaj içeriği ve FMI 2 FIFO'da depolanır.
- Eşleşme yoksa, gelen tanımlayıcı daha sonra maske modunda yapılandırılan filtrelerle karşılaştırılır.
- Tanımlayıcı, filtrelerde yapılandırılan tanımlayıcılardan hiçbiriyle eşleşmezse, mesaj, yazılımı rahatsız etmeden donanım tarafından atılır.

Mesaj Saklama

- Yazılım ve CAN mesajlarının donanımı arasındaki arayüz posta kutuları vasıtasıyla gerçekleştirilir. Bir posta kutusu, bir mesajla ilgili tüm bilgileri içerir; tanımlayıcı, veri, kontrol, durum ve zaman damgası bilgisi.

Transmit Mailbox

- Yazılım, boş bir posta kutusundan iletilecek mesajı ayarlar. İletimin durumu, CAN_TSR kaydındaki donanım ile gösterilir.

Offset to transmit mailbox base address (bytes)	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDHxR

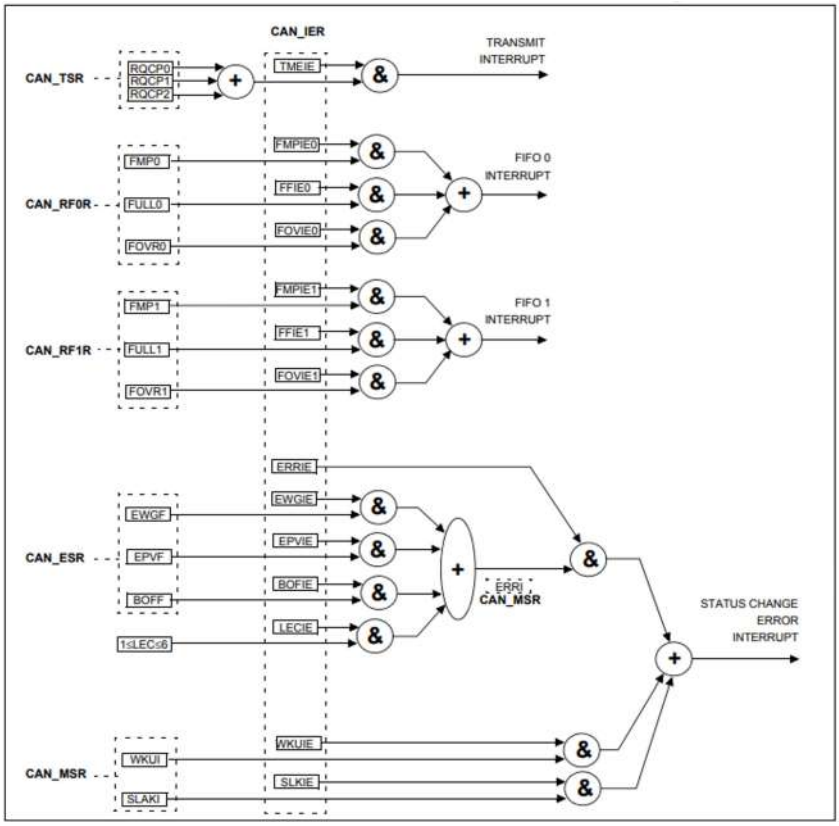
Receive Mailbox

- Bir mesaj alındığında, FIFO çıkış posta kutusundaki yazılım kullanılabilir.
- Yazılım mesajı ilettikten sonra (örneğin okuma) bir sonraki gelen mesajı mümkün kılmak için CAN_RFR kaydındaki RFOM biti aracılığıyla FIFO çıkış posta kutusunu serbest bırakmalıdır.
- Filtre eşleşme endeksi CAN_RDTXR yazmacının MFMI alanında saklanır.
- 16-bit zaman damgası değeri, CAN_RDTXR'nin TIME [15: 0] alanına kaydedilir.

Offset to receive mailbox base address (bytes)	Register name
0	CAN_RIxR
4	CAN_RDTxR
8	CAN_RDLxR
12	CAN_RDHxR

CAN Kesmeleri

- Dört kesme vektörü bxCAN'a tahsis edilmiştir.
- Her kesme kaynağı, CAN Kesme Kesintisi Kaydı (CAN_IER) aracılığıyla bağımsız olarak etkinleştirilebilir veya devre dışı bırakılabilir.



- Aktarım kesintisi aşağıdaki olaylar tarafından oluşturulabilir:
  - Gönderme posta kutusu 0 boş, CAN_TSR kayıt setinde RQCPO bit.

- Gönderilen posta kutusu 1 boşalır, CAN_TSR kayıt kümesinde RQCP1 biti.
- Gönderme posta kutusu 2 boşalır, CAN_TSR kayıt kümesinde RQCP2 biti.

- FIFO 0 kesmesi aşağıdaki olaylar tarafından oluşturulabilir:
  - Yeni bir mesaj alımı, CAN_RFOR yazmacındaki FMPO bitleri '00' değildir.
  - FIFOO tam koşulu, CAN_RFOR kayıt kümesinde FULLO biti.
  - FIFOO aşırı çalışma koşulu, CAN_RFOR kayıt setinde FOVRO biti.
- FIFO 1 kesmesi aşağıdaki olaylar tarafından oluşturulabilir:
  - Yeni bir mesaj alımı, CAN_RF1R yazmacındaki FMP1 bitleri '00' değildir.
  - FIFO1 tam koşulu, CAN_RF1R kayıt kümesinde FULL1 biti.
  - FIFO1 aşırı çalışma koşulu, CAN_RF1R kayıt setinde FOVR1 biti.
- Hata ve durum değişikliği kesintisi aşağıdaki olaylar tarafından oluşturulabilir:
  - Hata durumu, hata koşulları hakkında daha fazla bilgi için CAN Error Status register'a (CAN_ESR) bakın.
  - Uyandırma durumu, SOF, CAN Rx sinyalinde izlenir.
  - Uyku moduna giriş.

Register

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	CAN_MCR	Reserved															DBF	RESET	Reserved							TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
	Reset value																1	0								0	0	0	0	0	0	0	0
0x004	CAN_MSR	Reserved																				RX	SAMP	RXM	TXM	Res.			SLAKI	WKUI	ERRI	SLAK	INAK
	Reset value																					1	1	0	0				0	0	0	1	0
0x008	CAN_TSR	LOW[2:0]			TME[2:0]			CODE[1:0]		ABRQ2	Res.			TERR2	ALST2	TXOK2	RQCP2	ABRQ1	Res.			TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Res..			TERR0	ALST0	TXOK0	RQCP0
	Reset value	0	0	0	1	1	1	0	0	0				0	0	0	0	0				0	0	0	0	0				0	0	0	0
0x00C	CAN_RF0R	Reserved																									RFOM0	FOVR0	FULL0	Reserved	FMP0[1:0]		
	Reset value																										0	0	0		0	0	
0x010	CAN_RF1R	Reserved																									RFOM1	FOVR1	FULL1	Reserved	FMP1[1:0]		
	Reset value																										0	0	0		0	0	
0x014	CAN_IER	Reserved															SLKIE	WKUIE	ERRIE	Res.		LECIE	BOFIE	EPVIE	EWGIE	Reserved	FOVIE1	FFIE1	FMPIE1	FOVIE0	FFIE0	FMPIE0	TMEIE
	Reset value																0	0	0			0	0	0	0		0	0	0	0	0	0	0
0x018	CAN_ESR	REC[7:0]								TEC[7:0]									Reserved								LEC[2:0]			Reserved	BOFF	EPVF	EWGF
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										0	0	0		0	0	0
0x01C	CAN_BTR	SILM	LBKM	Reserved				SJW[1:0]		Reserved	TS2[2:0]			TS1[3:0]					Reserved							BRP[9:0]							
	Reset value	0	0					0	0		0	1	0	0	0	1	1									0	0	0	0	0	0	0	0
0x180	CAN_TI0R	STID[10:0]/EXID[28:18]												EXID[17:0]																	IDE	RTR	TXRQ
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0
0x184	CAN_TDT0R	TIME[15:0]																	Reserved						TGT	Reserved				DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								x					x	x	x	x
0x188	CAN_TDL0R	DATA3[7:0]								DATA2[7:0]									DATA1[7:0]							DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x18C	CAN_TDH0R	DATA7[7:0]								DATA6[7:0]									DATA5[7:0]							DATA4[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1B0	CAN_RI0R	STID[10:0]/EXID[28:18]												EXID[17:0]																	IDE	RTR	Reserved
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
0x1B4	CAN_RDT0R	TIME[15:0]																	FMI[7:0]							Reserved				DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x
0x1B8	CAN_RDL0R	DATA3[7:0]								DATA2[7:0]									DATA1[7:0]							DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1BC	CAN_RDH0R	DATA7[7:0]								DATA6[7:0]									DATA5[7:0]							DATA4[7:0]							

0x1B8																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1BC	CAN_RDH0R	DATA7[7:0]								DATA6[7:0]								DATA5[7:0]								DATA4[7:0]					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x200	CAN_FMR	Reserved																CAN2SB[5:0]						Reserved							FINIT
	Reset value																	0	0	1	1	1	0								1
0x204	CAN_FM1R	Reserved								FBM[27:0]																					
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20C	CAN_FS1R	Reserved								FSC[27:0]																					
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x214	CAN_FFA1R	Reserved								FFA[27:0]																					
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x21C	CAN_FA1R	Reserved								FACT[27:0]																					
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x240	CAN_F0R1	FB[31:0]																													
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
. . .	. . .	. . .																													
0x31C	CAN_F27R2	FB[31:0]																													
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

- **CAN Control and Status Registers**
  - **CAN_MCR** (Master Control Register), modülünün genel kontrolünü sağlar. Başlatma, uyku modu ve bus-off durumlarını yönetir.
    - **INRQ** (Initialization Request), modülünü başlatma moduna almak için kullanılır.
    - **SLEEP** (Sleep Mode Request), modülünü uyku moduna geçirmek için.
    - **TXFP** (Transmit FIFO Priority), Gönderim önceliğini ayarlar.
    - **ABOM** (Automatic Bus-Off Management), Otomatik bus-off yönetimini etkinleştirir.
  - **CAN_MSR** (Master Status Register), CAN modülünün durum bilgilerini verir. Modülün başlatma veya uyku modunda olup olmadığını ve hata durumlarını izler.
    - **INAK** (Initialization Acknowledge),Modülün başlatma modunda olduğunu belirtir.
    - **SLAK** (Sleep Acknowledge), Modülün uyku modunda olduğunu belirtir.
    - **ERRI** (Error Interrupt), Bir hata kesmesi meydana geldiğini gösterir.
    - **WKUI** (Wakeup Interrupt), Modülün uyandırıldığını belirtir.
  - **CAN_TSR** (Transmit Status Register), Mesaj gönderimiyle ilgili durumu raporlar.
    - **RQCPx** (Request Completed x), Her bir iletim kutusundaki (mailbox) iletimin tamamlandığını gösterir.
    - **TME** (Transmit Mailbox Empty), İletim kutularının boş olduğunu belirtir.
    - **ABRQx** (Abort Request x), İlgili iletim kutusundaki mesajın iptal edilmesini sağlar.
  - **CAN_RFxR** (Receive FIFO Register), Alım FIFO'larının durum bilgilerini ve kontrolünü sağlar. FIFO'daki mesaj sayısı, FIFO taşması ve FIFO mesaj boşaltma işlemlerini izler.
    - **FMPx** (FIFO Message Pending x), FIFO'da bekleyen mesaj sayısını gösterir.
    - **FULLx** (FIFO Full x), FIFO'nun dolu olup olmadığını belirtir.
    - **FOVRx** (FIFO Overrun x), FIFO taşması meydana geldiğinde set edilir.
    - **RFOMx** (Release FIFO Output Mailbox x), FIFO'dan bir mesajın serbest bırakılmasını sağlar.
  - **CAN_IER** (Interrupt Enable Register), CAN modülündeki kesme kaynaklarını etkinleştirir.
    - **TMEIE** (Transmit Mailbox Empty Interrupt Enable), İletim kutusu boş olduğunda kesmeyi etkinleştirir.
    - **FMPIEx** (FIFO Message Pending Interrupt Enable), FIFO'ya mesaj geldiğinde kesmeyi etkinleştirir.
    - **ERRIE** (Error Interrupt Enable), Hata kesmesini etkinleştirir.
  - **CAN_ESR** (Error Status Register), CAN modülündeki hata durumlarını izler. Hata uyarı bayrakları, bus-off durumu ve son hata kodu gibi bilgileri içerir.
    - **EWGF** (Error Warning Flag), Hata durumu uyarı bayrağı.
    - **BOFF** (Bus-Off Flag), Bus-off durumunu gösterir.
    - **LEC** (Last Error Code**)**, Son hata kodunu belirtir (bit hata, CRC hata vb.).
  - **CAN_BTR** (Bit Timing Register), CAN bit zamanlaması ve baud rate ayarlarını yapar. Yeniden senkronizasyon genişliği (SJW), zamanlama segmentleri (TS1 ve TS2) ve prescaler (BRP) bu registerda yer alır.

    - **CAN_BTR** (Bit Timing Register), CAN bit zamanlaması ve baud rate ayarlarını yapar. Yeniden senkronizasyon genişliği (SJW), zamanlama segmentleri (TS1 ve TS2) ve prescaler (BRP) bu registerda yer alır.
        - **BRP** (Baud Rate Prescaler), CAN hızını belirlemek için prescaler değeri.
        - **TS1** (Time Segment 1), Zamanlama segmenti 1.
        - **TS2** (Time Segment 2), Zamanlama segmenti 2.
        - **SJW** (Synchronization Jump Width), Yeniden senkronizasyon genişliği.
- **CAN Mailbox Register**: Posta kutuları, mesaj gönderimi ve alımı için kullanılan yapılar olup aşağıdaki registerlardan oluşur.
    - **CAN_TIxR ve CAN_RIxR** (Transmit/Receive Identifier Register)**,** gönderilecek/alınan mesajın kimlik bilgisini içerir.
        - **STID** (Standard Identifier), 11 bit Standard mesaj kimliğini içerir
        - **EXID** (Extended Identifier), 29 bit Extended mesaj kimliği için kullanılır
        - **IDE** (Identifier Extension), mesaj kimliği tipini seçer. Standard ya da Extended seçimi yapılır.
        - **RTR** (Remote Transmission Request), Mesaj tipini belirtir. Data frame ya da Remote frame seçimi yapılır.
        - **TXRQ** (Transmit mailbox request), mesajının gönderilmek üzere hazır olduğunu gösteren bir kontroldür. ilgili posta kutusu için iletimi talep etmek üzere yazılım tarafından ayarlanır. Posta kutusu boşaldığında donanım tarafından temizlenir
    - **CAN_TDTxR ve CAN_RDTxR** (Transmit/Receive Data Length Control and Time Stamp Register)**,** gönderilecek/alınan mesajın veri uzunluğunu ve isteğe bağlı zaman damgasını içerir
        - **DLC** (Data Length Code), Gönderilecek veri uzunluğunu belirtir (0-8 byte)
        - **TGT** (Transmit Global Time), zaman damgası kullanımı için ayar
        - **TIME** (Time Stamp), opsiyonel olarak zaman damgası değeri içerir.
    - **CAN_TDLxR, CAN_TDHxR, CAN_RDLxR ve CAN_RDHxR (**Transmit/Receive Data Low/High Register), gönderilecek/alınan mesajın düşük (ilk 4 byte) ve yüksek (son 4 byte) veri kısmını içerir.
        - **DATA**, Alınan mesajın veri içeriğini tutar
- **CAN Filter Registers:** Filtreler, CAN modülüne ulaşan mesajları filtrelemek için kullanılır. Aşağıdaki registerlar filtrelerin kontrolünü sağlar.
    - **CAN_FMR,** (Filter Master Register), filtrelerin genel kontrolünü sağlar.
        - **FINIT (Filter Initialization Mode)**: Filtrelerin yapılandırılmasını sağlamak için kullanılır
    - **CAN_FM1R** (Filter Mode Register), filtrelerin modunu ayarlar. Her bir filtre bankası için bir bit ayrılmıştır ve her bit için **Mask mode** veya **List mode** seçimi yapılır.
    - **CAN_FS1R** (Filter Scale Register), filtrelerin ölçeğini ayarlar. Her bir filtre bankası için bir bit ayrılmıştır ve her bit için **16 bit** veya **32 bit** seçimi yapılır.
    - **CAN_FFA1R** (Filter FIFO Assignment Register), filtrelerin hangi FIFO'ya yönlendirilmesi gerektiğini belirler. Her bir filtre bankası için bir bit ayrılmıştır ve her bit için **FIFO0** veya **FIFO1** seçimi yapılır.
    - **CAN_FA1R** (Filter Activation Register), filtrelerin etkin olup olmadığını kontrol eder. Her bir filtre bankası için bir bit ayrılmıştır ve her bit için **devre dışı** veya **etkin** seçimi yapılır.
    - **CAN_FiRx** (Filter Bank Registers), her filtre bankasının (CAN_F0R1, CAN_F0R2, …, CAN_F27R1, CAN_F27R2) **kimlik** ve **maske** bilgilerini içerir. İlgili bitlere bağlı olarak gelen mesajlar filtrelenir.