

# STM32 ile Gömülü Yazılım

5 Mayıs 2021 Çarşamba 07:50

[Giriş](#)

[01 GPIO](#)

[02 EXTI](#)

[03 ADC](#)

[04 DAC](#)

[05 DMA](#)

[06 TIMER](#)

[07 PWM](#)

[08 UART](#)

[09 SPI](#)

[10 I2C](#)

[Uygulamalar](#)

[Harici Led Yakma](#) [HAL, REGISTER]

[Buton ile Led Yakma](#) [HAL, REGISTER]

[External Interrupt](#) [HAL, REGISTER]

[ADC Verisi Okuma](#) [HAL, REGISTER]

[ADC Interrupt](#) [HAL]

[DAC Kullanımı](#) [HAL, REGISTER]

[ADC Değeri İle DAC Kontrolü](#) [HAL]

[DMA ile ADC Değer Okuma](#) [HAL, REGISTER]

[Timer Değer Okuma](#) [REGISTER]

[Timer Interrupt](#) [HAL]

[Timer ile Delay Oluşturma](#) [HAL]

[PWM Kullanımı](#) [HAL, REGISTER]

[USART ile Mesaj Gonderme](#) [HAL, REGISTER]

[SPI Kullanımı](#) [HAL]

[I2C Kullanımı](#) [REGISTER]

# Giriş

5 Mayıs 2021 Çarşamba 08:02

## Giriş

### Kaynaklar

- Bu belge oluşturulurken <https://www.udemy.com/course/stm32f4-discovery-kart-ile-arm-dersleri/> linkteki eğitim kursu izlenirken alınan notlardan oluşmaktadır.

### Giriş

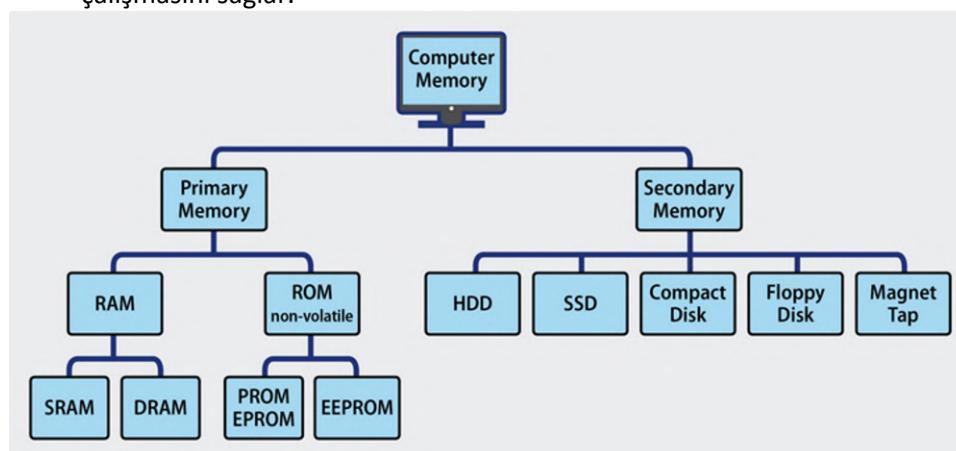
- <https://www.elektrikport.com/universite/gomulu-sistem-nedir/8658#ad-image-0> ile <https://maker.robotistan.com/mikroislemci/> linkteki Gömülü Sistem, Mikroişlemci, Mikrodenetleyici nedir sorularına cevap veren yazıları okuyabilirsiniz.

### Mikroişlemci

- Mikroişlemci yapısında bir CPU (Central Processing Unit), ön bellek ve input/output (giriş/ Çıkış) birimleri bulunan devrelere microprocessor denir.
- Bu üç temel unsur birbirlerine bus, iletişim yolları ile bağlıdır.
- Mikroişlemcinin beyni CPU' dur. Veri akışı ve veri işleme bu birim sayesinde gerçekleşir.
- Bu veri işleme genellikle CPU içerisinde yer alan ALU (Aritmetic Logic Unit)' da uygulanır. Bu birimde sayısal ve lojik işlemler yapılır. Tüm dijital elektronik işlemler CPU ların en temel işlemleridir.
- CPU'ların içerisinde 8-16-32-64 bitlik registerler bulunmaktadır. Register, bilgilerin geçici sürede depolanmasını sağlarlar.
- CPU' lar, mikroişlemcinin hafızasındaki programları bulma, çağrıma ve onları çalıştırma görevi görürler. Veri İşleme Adımları:; Veriyi Getirmek (Fetch), Veriyi Çözmek (Dekode), Veriyi İşlemek (Execute), Veriyi Hafızata, Geri Depolamak (Store)
- Merkezi işlem birimi üç birimden oluşur.

**ALU**, hafıza biriminden gelen verilerin işlenmesinde görev alır. Bu işlemlerise aritmetik olarak toplama, çıkarma, bölme ve çapmadır. İkili sayı tabanındaki (binary) mantık işlemleri VE (AND), OR (VEYA) ve bit kaydırma işlemleridir.

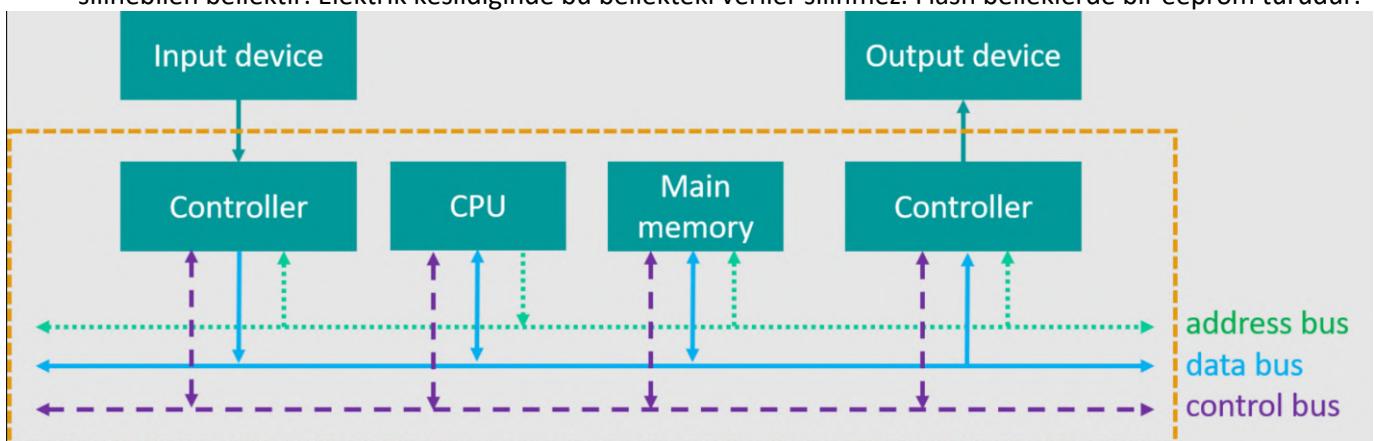
**REGISTER**, hafızadaki veriler ALU tarafından işlenirken kullanılan geçici ve kalıcı saklayıcılardır. Registerler işlemcinin çekirdeğinde olduklarından verilere ulaşmak daha hızlı şekilde 3- Control Unit: Kontrol birimi, işlemcinin çalışmasını yönlendiren birimdir. İşlemci içerisindeki ve dışarısındaki birimlerin senkron şekilde çalışmasını sağlar.



- RAM, ROM ve EEPROM hafızasının temel birimleridir.
- Mikroişlemciye atılan veriler ilk olarak hafızaya gelir ve burada depolanır. CPU'ların doğrudan eriştiği birim bellektir. Bellekte iki tane birincil hafıza birimi vardır.
- RAM (Random Access Memory)**, mikroişlemcinin elektrik alması durumunda geçici hafıza olarak kullandığı birimdir. Elektrik kesildiği zaman bu veriler silinir ve bir daha kullanılmaz. RAM, diğer hafıza birimleri gibi verileri önceden verilen bir sırayla dizmez. Bu sebeple ismi rastgele erişim bellek olarak konulmuştur. RAM, dinamik Rastgele Erişim Bellek ve Statik Rastgele Erişim Bellek olmak üzere ikiye ayrılır.
- ROM (Read Only Memory)**, sadece okunabilir bir bellektir. Elektrik kesildiğinde bu bellekteki veriler silinmez.

ROM üzerindeki yazılmış fabrikasyon yazılımlar kullanıcılar tarafından değiştirilebilir, silinemez.

- **EEPROM (Electrically Erasable Programmable Read-Only Memory)**, elektrik ile defalarca yazılıp silinebilen bellektir. Elektrik kesildiğinde bu bellekteki veriler silinmez. Flash belleklerde bir eeprom türüdür.



- Giriş - Çıkış birimleri mikroişlemci ile dış dünyadan sinyaller aracılığı ile haberleştiği birimdir.
- Bu giriş ve çıkışlar; giriş/ Çıkış portları, harici elektronik birimler, fiziksel cihazlar ve yazılımlar olabilir.
- CPU daki veri akışının aktarılması, bellek ve giriş/çışı birimlerinin bağlantılarını sağlayan üç çeşit bus vardır.

**Address Bus**, verinin okunacağı veya verinin yazılacağı bölgeyi belirten adres bilgilerinin taşınmasını sağlar. Tek yönlü bir veri yoludur.

**Data Bus**, CPU dan bellek ve giriş/ Çıkış portlarına veya bu birimlerden CPU' ya çift yönlü bir hat vardır.

**Control Bus**, Mikroişlemcideki birimler arasında iletişimini sağlayan sinyalleri iletan, kontrol eden veri hattıdır. Her mikroişlemci farklı sayıda control bus'a sahiptir.

## Mikrodenetleyici

- Mikroişlemcili bir sistemin içerisinde bulunması gereken temel bileşenlerden RAM, ROM, ALU, kontrol ünitesi ve I/O ünitesini tek bir çip içerisinde barındıran entegre devreye microcontroller denir.
- Mikrodenetleyici, dışarıdan gelen bir veriyi hafızasına alan, derleyen ve sonucunda çıktı elde eden bir bilgisayardır. Mikrodenetleyicilerin yapısında; CPU, RAM, ROM, I/O Portları, Seri ve Paralel Portlar, Zamanlayıcılar, ADC ve DAC çevre birimleri
- Mikrodenetleyiciler gerçek zamanlı (real time) işlemlerde oldukça başarılılardır.
- Mikrodenetleyiciler herhangi bir işi çok küçük boyutlarda ve daha düşük enerjide yaparlar.
- Mikroişlemcili ile kontrol edilecek bir sistemi kurmak için gerekli olan minimum donanımda CPU, RAM, I/O bulunmalıdır. Bunlar arasında veri alışverişini sağlamak için ise veri yolu, adres yolu ve kontrol yolu gereklidir. Birimler arasındaki iletişimini sağlayan bu yolları yerleştirmek içinde bir anakart gereklidir.
- Mikrodenetleyici ile kontrol edilecek sistemde ise yukarıda söylediğimiz birimler tek zaten mikrodenetleyici içerisinde bulunmaktadır. Bu da maliyetin daha düşük olacağı anlamına gelir.
- Mikrodenetleyiciler çok az sayıda ve karmaşık olmayan komutlarla programlanabilen sistemlerdir.
- Mikronetleyiciler hız bakımından mikroişlemcilerden daha hızlıdır. Güç tüketimi mikrodenetleyicilerde daha azdır. Fiyatları mikroişlemcilere göre daha uygundur.
- Mikroişlemcili sistemlerde harici donanım desteği gerekliliği iken, mikrodenetleyicilerde bu gereklilik çok azdır.

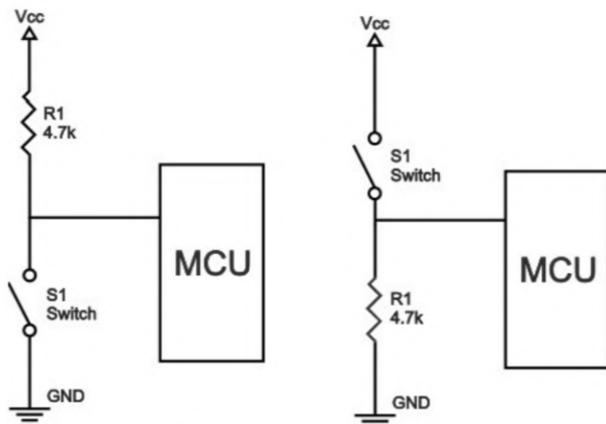
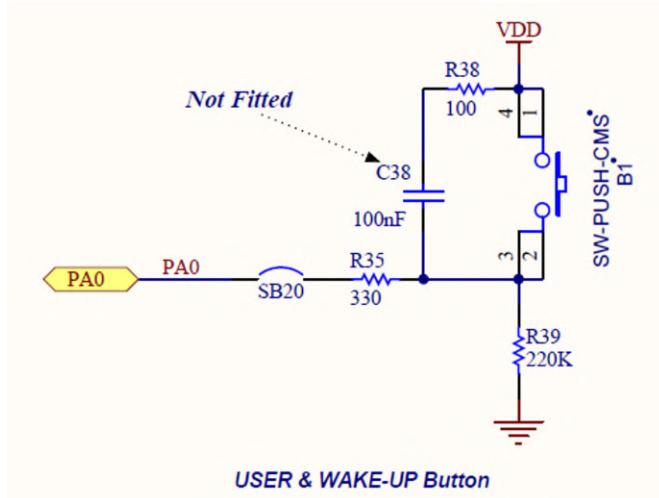
# 01 GPIO

5 Mayıs 2021 Çarşamba 08:02

## 01 GPIO

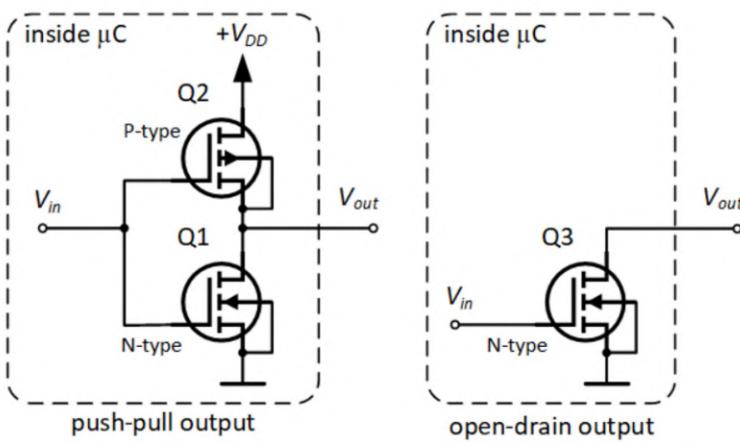
### Giriş

- Butonlar ve anahtarlar mikrodenetleyiciye giriş pini üzerinden lojik 1 ve lojik 0 olarak bilgi girişini sağlayan mekanik elemanlardır.
- Resimde görüldüğü gibi kullanıcı butonu A portunun 0. pinine bağlı ve pull down durumundadır.

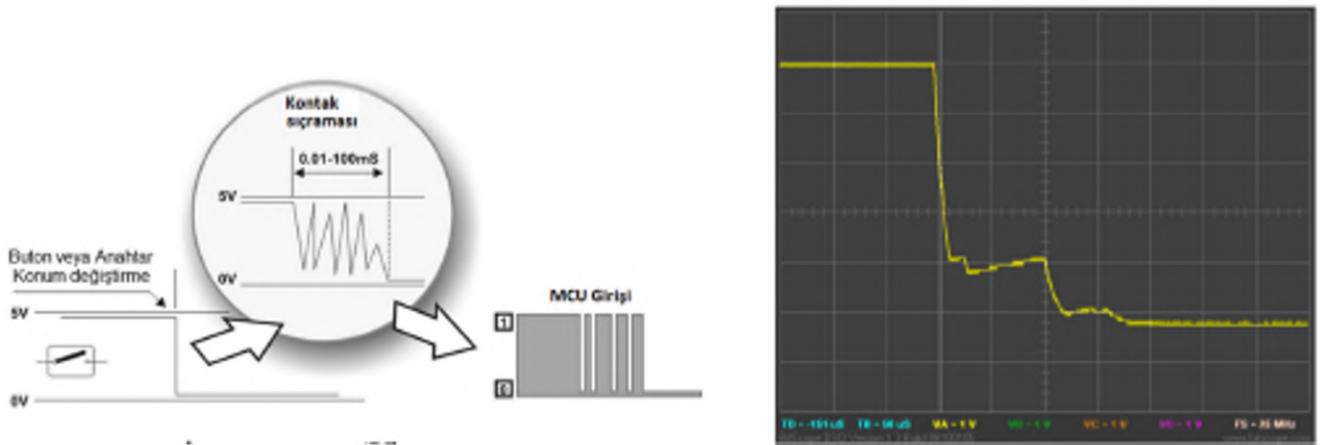


### Pull-Up Direnç

- PullUp bağlantıda GPIO girişi direnç üzerinden + beslemeye (VCC/VDD) bağlanır. Butona basılmadığı durumda GPIO girişinde lojik 1 vardır. Butona basıldığı durumda girişe 0V (lojik 0) uygulanmış olur.
- PullDown bağlantıda, GPIO girişi direnç üzerinden GND ye bağlanır. Butona basılmadığı durumda girişte lojik 0 bulunur. Butona basıldığı durumda buton üzerinden lojik 1 uygulanmış olur.

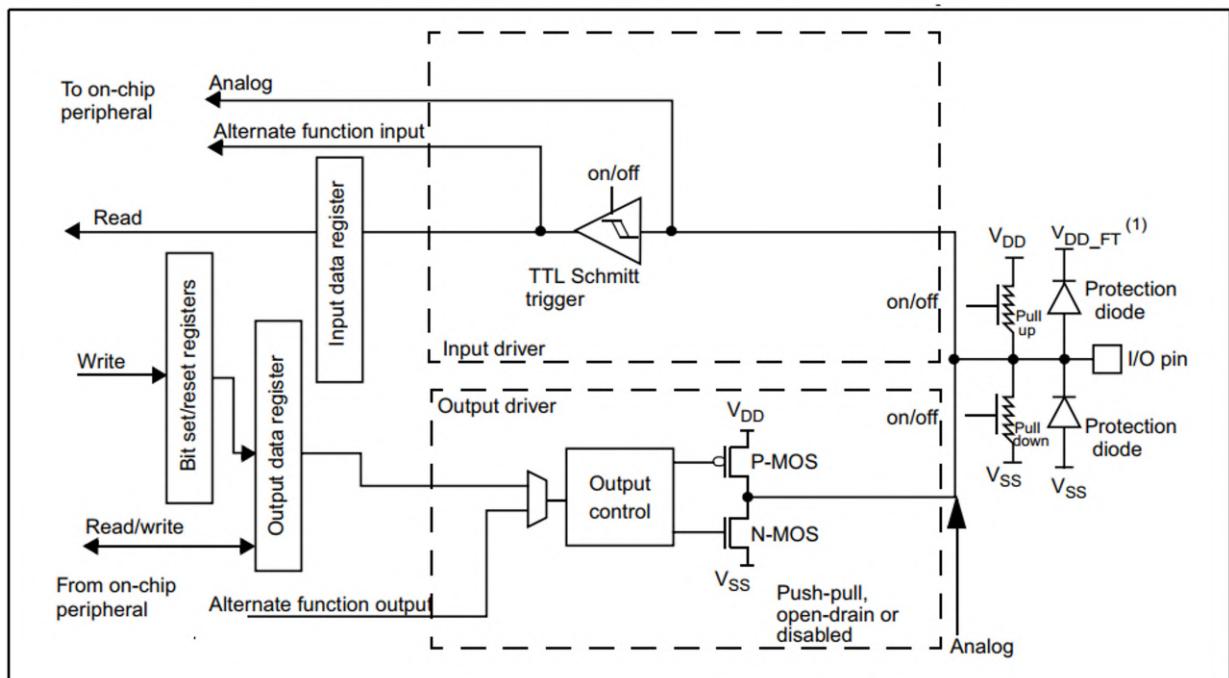


- Buton ve anahtarda konum değiştiğinde arktan dolayı mikrodenetleyici girişinde çok sayıda istenmeyen lojik değer oluşur. Bu duruma ark deniyor.



- Ark problemini <https://akademi.robolinkmarket.com/buton-arki-nedir-nasıl-cozulur/> linkten donanımsal ve yazılımsal olarak paylaşılan çözümleri inceleyip uygulayabiliriz.

## Birim Yapısı



## Register

- **GPIOx\_MODER (Mode Register)**, her pin için iki bit kullanılır. Giriş, çıkış, alternatif fonksiyon veya analog modunu seçmek için kullanılır.
  - **GPIOx\_OTYPER (Output Type Register)**, her pin için bir bit kullanılır. Push-pull veya Open-drain çıkış tipini seçmek için kullanılır.
  - **GPIOx\_OSPEEDR (Output Speed Register)**, her pin için iki bit kullanılır. Çıkış hızını kontrol etmek için kullanılır.
  - **GPIOx\_PUPDR (Pull-up/Pull-down Register)**, her pin için iki bit kullanılır. Dahili pull-up veya pull-down direncini etkinleştirmek için kullanılır.
  - **GPIOx\_IDR (Input Data Register)**, her pin için bir bit kullanılır. Pinin mevcut durumunu okumak için kullanılır.
  - **GPIOx\_ODR (Output Data Register)**, her pin için bir bit kullanılır. Çıkış durumunu ayarlamak veya temizlemek için kullanılır.
  - **GPIOx\_BSRR (Bit Set/Reset Register)**, her pin için iki bit içerir. Bir GPIO pininin durumunu set etmek veya resetlemek için kullanılır.
  - **GPIOx\_LCKR (Lock Register)**, her pin için bir bit içerir. GPIO pin konfigürasyonunun kilitlenmesini sağlar.

- **GPIOx\_AFRL** ve **GPIOx\_AFRH (Alternate Function Low/High Register)**, her biri 32-bit uzunluğunda iki registerdir ve her pin için dört bit içerir. GPIO pinlerinin alternatif fonksiyonlarını belirlemek için kullanılır.

## Kontrol Yöntemleri

- GPIO pinlerini kontrol etmek için iki temel yöntem vardır. Bunlar interrupt ve polling. İşlemcinin ve uygulamanın gereksinimlerine bağlı olarak her iki yöntem de tercih edilebilir.
- **Polling yöntemi**, mikrodenetleyici tarafından belirli bir durumun sürekli olarak kontrol edilmesine dayanır. Örneğin, bir GPIO pininin durumu sürekli bir döngü içinde kontrol edilebilir. Avantajları basit ve doğrudan bir yaklaşım ile donanım ve yazılım karmaşıklığı düşüktür. Dezavantajları sürekli olarak işlem yaparak sistem kaynaklarını tüketir. Anında tepki verme yeteneği sınırlıdır.  
Basit uygulamalarda veya sürekli düşük güç tüketimi gerektiren durumlarda tercih edilebilir. Kesmelerin işlemi engelleyeceği veya karmaşık hale getireceği durumlarda kullanılmalıdır. Zamanlama veya hızlı tepki gerekliliğinde kullanılabilir.
- **Interrupt yöntemi**, bir olay (örneğin, GPIO pininin durum değiştirmesi) gerçekleştiğinde normal programın çalışmasını kesip belirli bir kesme servis rutinini çalıştırarak olaya tepki verir. Avantajları düşük enerji tüketimi, çünkü işlemci, beklenmeyen olaylar olana kadar bekler. Anında tepki verme yeteneği yüksektir.  
Dezavantajları, Kod karmaşıklığı ve debug işlemleri artabilir. Zamanlaması hassas olabilir ve bazı durumlarda kesmeler birbirini engelleyebilir.  
Anında tepki gerektiren durumlarda (örneğin, düğme basıldığında). Enerji tüketiminin daha fazla toleranslı olduğu durumlarda. Sık sık kontrol etmenin pratik olmadığı durumlar için uygun bir seçenekdir.
- Genel olarak, interrupt yöntemi, enerji tüketimi veya anında tepki gereksinimleri gibi durumlarda daha uygun olabilirken, sadece belirli durumlarda kontrol yapılması gereken basit uygulamalarda polling sorgulama kullanılabilir.

# 02 EXTI

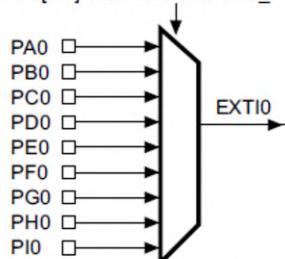
5 Mayıs 2021 Çarşamba 08:02

## 02 EXTI

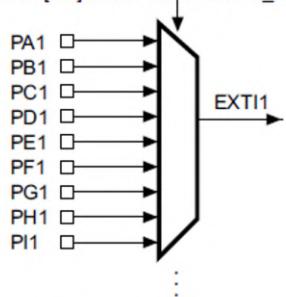
### Giriş

- Önceliği yüksek işlerin mikrodenetleyici tarafından ana program akışını keserek yapılmasına interrupt denir.
- Eğer bir kesme kaynağından mikrodenetleyiciye uyarı gelirse mikrodenetleyici yapmakta olduğu işi bekletir, kesme alt programına gider, o programı icra eder, daha sonra ana programda kaldığı yerden devam eder.
- Kesmeleri genellikle çok hızlı yapılması gereken işlemlerde, anlık tepki verilmesi gereken yerlerde kullanırız.
- Harici bir kaynaktan oluşan olaylardan dolayı meydana gelen kesmelere, harici kesmeler denir.  
Harici kaynak olarak, dış ortamdan pinler vasıtasyyla gelecek kesme ve kandi içindeki donanımlardan gelen kesmeleri anlayabiliriz.
- STM32F407 mikrodenetleyicisi için porttaki 0.pin EXTI0 kanalına bağlıdır.

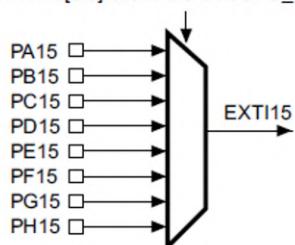
EXTI0[3:0] bits in the SYSCFG\_EXTICR1 register



EXTI1[3:0] bits in the SYSCFG\_EXTICR1 register



EXTI15[3:0] bits in the SYSCFG\_EXTICR4 register



- Bunlar dışında 7 tane daha kanal vardır. Toplamda 23 kanal vardır.

EXTI line 16 is connected to the PVD output

EXTI line 17 is connected to the RTC Alarm event

EXTI line 18 is connected to the USB OTG FS Wakeup event

EXTI line 19 is connected to the Ethernet Wakeup event

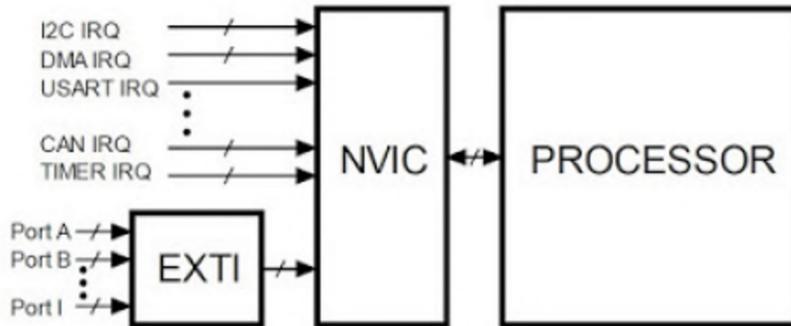
EXTI line 20 is connected to the USB OTG HS (configured in FS) Wakeup event

EXTI line 21 is connected to the RTC Tamper and TimeStamp events

EXTI line 22 is connected to the RTC Wakeup event

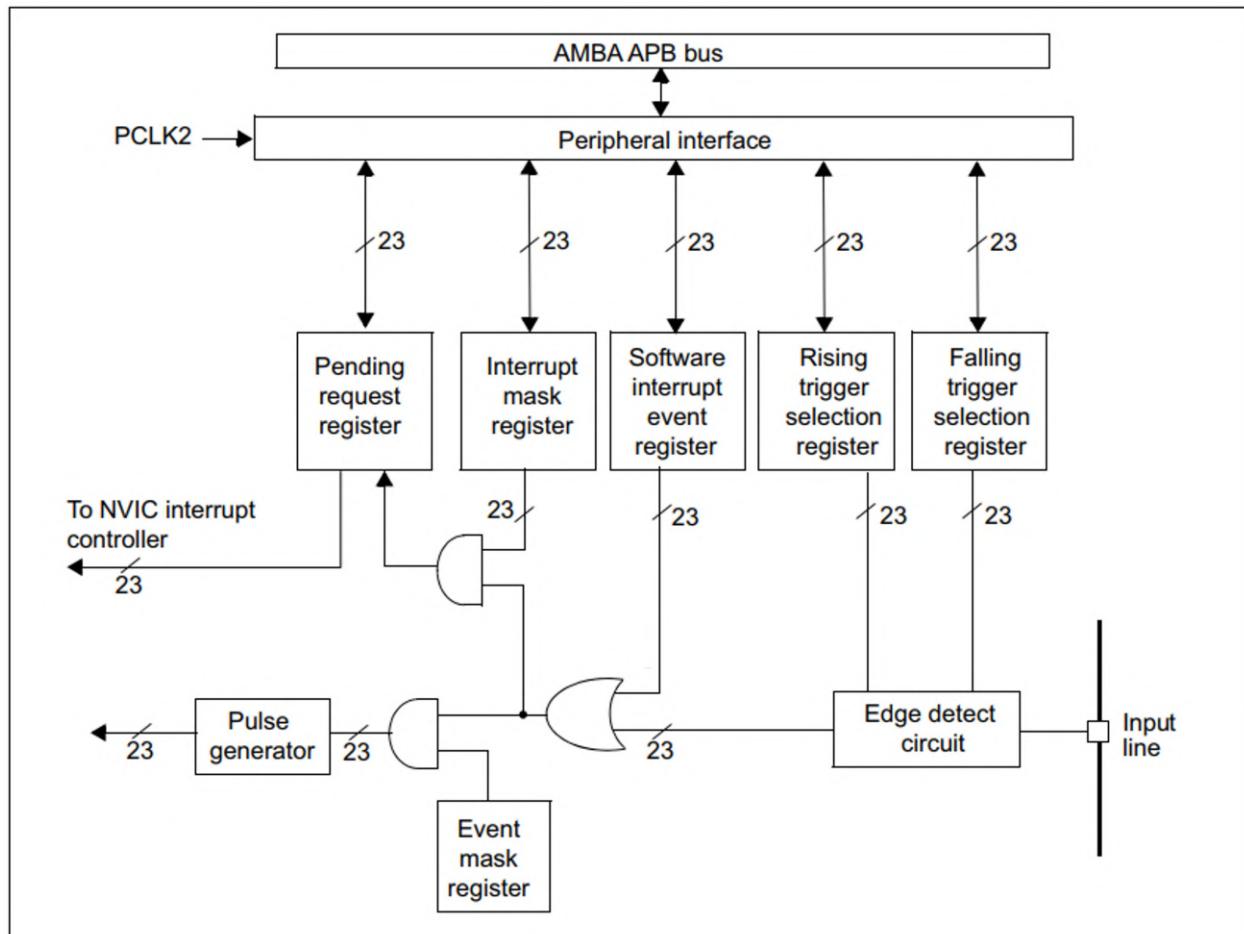
- Karmaşık kesme isteklerinin işlemciye sürekli yük getirmemesi için işlemci içerisinde özel bir donanım bloğu oluşturulmuştur. Bu donanıma interrupt controller adı verilir.
- Kesme kontrolörü haklı bir sebeple gelen kesme isteği neticesinde düzgün işleyen programı askıya alarak kesme fonksiyonu (interrupt function) olarak adlandırılan özel kod parçasını işlemeye başlar.
- Kesme fonksiyonunun işletilmesinin bitiminde program kaldığı yerden çalışmaya devam eder.
- NVIC kontrolör mikroişlemci içerisindeki önemli donanım kesmelerini (DMA, USART, CAN, I2C ve Timer gibi)

ve ayrıca External Interrupt (EXTI) adı verilen donanım vasıtasyyla portlardan gelen kesmeleri kontrol eder.



- Interrupt kullanmak için üç farklı yapıyı ayarlamak gerekiyor. SYSCFG, EXTI ve NVIC yapılarını ayarlanarak interrupt kullanabilirim.

## Birim Yapısı



## Register

- **SYSCFG\_MEMRMP (Memory Remap Register)**, mikrodenetleyicinin bellek haritalamasını yapılandırmak için kullanılır. Bellek haritalaması, sistemdeki farklı bellek alanları arasındaki bağlantıları yönetir. Örneğin, boot sektörünü değiştirmek veya haritalamayı farklı bir bellek bölgесine taşımak için kullanılabilir.
  - **SYSCFG\_PMC (Peripheral Mode Configuration Register)**, çeşitli periferiklerin davranışlarını yapılandırmak için kullanılır. Özellikle çeşitli periferiklerin hangi güç modunda çalışacaklarını belirlemek için kullanılır.
  - **SYSCFG\_EXTICR (External Interrupt Configuration Registers)**, harici kesmelerin hangi pinlere bağlı olduğunu yapılandırmak için kullanılır. Genellikle harici donanım kesmelerini bir GPIO pinine atanabilir ve bu registerlar aracılığıyla bu atamalar yapılır.
  - **SYSCFG\_CMPCR (Compensation Cell Control Register)**, gerilim takibi ve düzeltme için kullanılır. Gerilim takibi, mikrodenetleyicinin çalışma gerilimini izleyerek enerji verimliliğini artırabilir.

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
0x00	<b>EXTI_IMR</b> Reset value	Reserved										MR[22:0]																					
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x04	<b>EXTI_EMR</b> Reset value	Reserved										MR[22:0]																					
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x08	<b>EXTI_RTSR</b> Reset value	Reserved										TR[22:0]																					
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x0C	<b>EXTI_FTSR</b> Reset value	Reserved										TR[22:0]																					
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x10	<b>EXTI_SWIER</b> Reset value	Reserved										SWIER[22:0]																					
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x14	<b>EXTI_PR</b> Reset value	Reserved										PR[22:0]																					
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

- **EXTI\_IMR (Interrupt Mask Register)**, harici kesmelerin genel olarak etkinleştirilip etkinleştirilmeyeceğini kontrol eder. Her bit, belirli bir harici kesme hattını temsil eder ve bu bitlerin set olması, ilgili kesmenin etkinleştirildiği anlamına gelir.
- **EXTI\_EMR (Event Mask Register)**, EXTI modülü, hem kesme (interrupt) hem de event modlarında çalışabilir. Belirli bir harici kesme hattının olay modunda çalışıp çalışmayağını kontrol eder. Yine, her bit belirli bir kesme hattını temsil eder.
- **EXTI\_RTSR (Rising Trigger Selection Register)**, bir harici kesmenin hangi kenardan rising edge tetikleneceğini belirler. Her bit, bir kesme hattını temsil eder ve bu bitlerin set olması, ilgili kesmenin yükselen kenardan tetikleneceği anlamına gelir.
- **EXTI\_FTSR (Falling Trigger Selection Register)**, bir harici kesmenin hangi kenardan falling edge tetikleneceğini belirler. Yine, her bit bir kesme hattını temsil eder ve bu bitlerin set olması, ilgili kesmenin düşen kenardan tetikleneceği anlamına gelir.
- **EXTI\_SWIER (Software Interrupt Event Register)**, yazılımsal olarak bir harici kesme talebi oluşturmak için kullanılır. Her bit, belirli bir harici kesmeyi temsil eder ve bu bitin set olması, ilgili kesme hattına bir yazılımsal talep gönderileceği anlamına gelir.
- **EXTI\_PR (Pending Register)**, hangi harici kesmelerin beklediğini gösterir. Her bit, belirli bir kesme hattını temsil eder ve bu bitlerin set olması, ilgili kesmenin beklediği anlamına gelir. Yazılım tarafından temizlenmelidir.

# 03 ADC

5 Mayıs 2021 Çarşamba 08:02

## 03 ADC

### Giriş

- Doğada var olan bütün fiziksel büyüklükler (ısı, ışık, ses, zaman vs.) analog büyüklik kavramına girer.
- Dünyadaki herhangi bir şeyi dijital sistemlerimiz ile ölçmek, değerlendirmek, işlemek ve bu değerlere göre işlem yapabilmek için ADC (Analog Digital Converter) ihtiyaç vardır.
- ADC modülleri gerek harici, gerek dahili olsun hepsi bir referans voltajı ihtiyaç duyarlar. Genellikle mikroşlemcilerde referans voltajı işlemcinin besleme gerilimidir. Bu değer aynı zamanda ayarlar yapılarak harici olarak verilebilir.
- STM32'de 12-bit ADC, ardışık yaklaşım prensibine dayanan bir analog-dijital çeviricidir. Bu çevirici, 16 harici kaynaktan, iki dahili kaynaktan ve VBAT kanalından gelen sinyalleri ölçebilmek için en fazla 19 multiplexli kanala sahiptir. Kanalların A/D dönüşümü single, continuous, scan veya discontinuous modda gerçekleştirilebilir. ADC'nin sonucu, sola ya da sağa hizalanmış 16-bit veri kaydına depolanır.
- analog watchdog özelliği, uygulamanın giriş voltajının kullanıcı tanımlı üst veya alt sınırları aşmasını algılamasına olanak tanır.

### Çözünürlük

- ADC'ler 10, 12, 16, 24 vb. bit çözünürlükte bulunurlar.
- STM32F407'de ADC'ler 6, 8, 10 ve 12 bit çözünürlükte çalışabilirler.
- Referans voltajı default 3.3V'dur.
- ADC modülün 10 bit olduğunu düşünelim.  $2^{10} = 1024$  değeri okunacak maksimum değerdir yani  $0V=0$ ,  $3.3V=1023$  değeri bize döner. Buradan her bit değerinin alacağı voltaj değerini  $3.3 / 1024 = 0,0032$  olarak buluruz. Buradan da biz ADC modülünden okuduğumuz değeri bu ifade ile çarparsak voltaj değerini buluruz. 640 değeri için  $640 * 0,0032 = 2,048$  V olarak buluruz.
- STM32F407'de 0-3.6V aralığında ölçümler yapılmaktadır. Buradaki voltaj aralığında ADC birimin beslemesi (VDDA-VSSA) ile ilgili bir durumdur.
- ADC birimin besleme voltajı (VDD) ve referans gerilimi (VREF), ADC birimin ölçüleceği gerilim aralığını belirler.
- Her ne olursa olsun ADC birimi 3.6V'dan fazlasını ölçemez.
- Analog bir değerden dijital bir değer dönüşüm yapılırken dikkat edilmesi gereken hususlar vardır. Bunlardan en önemlisi, ölçülecek analog gerilim değerinin dönüşümü yapacak çipin **ölçüm aralığında** olması gereklidir. Diğer en önemli nokta, ölçüm yapılacak **hassasiyetin belirlenmesi** ve buna uygun bir genişliğinde bir dönüştürücü seçilmelidir.
- Ölçüm hassasiyetinde önemli olan dönüşüm yapacak sistemin bir çözünürlüğüdür.  
Resolution =  $VREF/(2^{n-1})$

Örneğin 0-3.3V aralığı arası ölçüm yapabilen bir ADC ölçüm ünitesinin ölçüleceği minimum değer yaklaşık olarak formülden 8 bit çözünürlük için 12mV, 12 bit çözünürlük için 805uV'tur.

### Çevrim Süresi

- <https://controllerstech.com/adc-conversion-time-frequency-calculation-in-stm32/> linkten ADC için çevrim süresinin nasıl hesaplandığı ile ilgili yazıyı okuyabiliriz.
- STM32F407'de ADC birimin ulaşabileceği maximum hız 36 MHz'dir. Bu hız aynı zamanda ADC çözünürlüğü ile ters orantılıdır. Çözünürlük arttıkça ADC birimin ölçüm hızı düşmektedir.

ÇÖZÜNÜRLÜK	ADC ÇEVİRİM HIZI
12 Bit	12 Cycle
10 Bit	10 Cycle
8 Bit	8 Cycle
6 Bit	6 Cycle

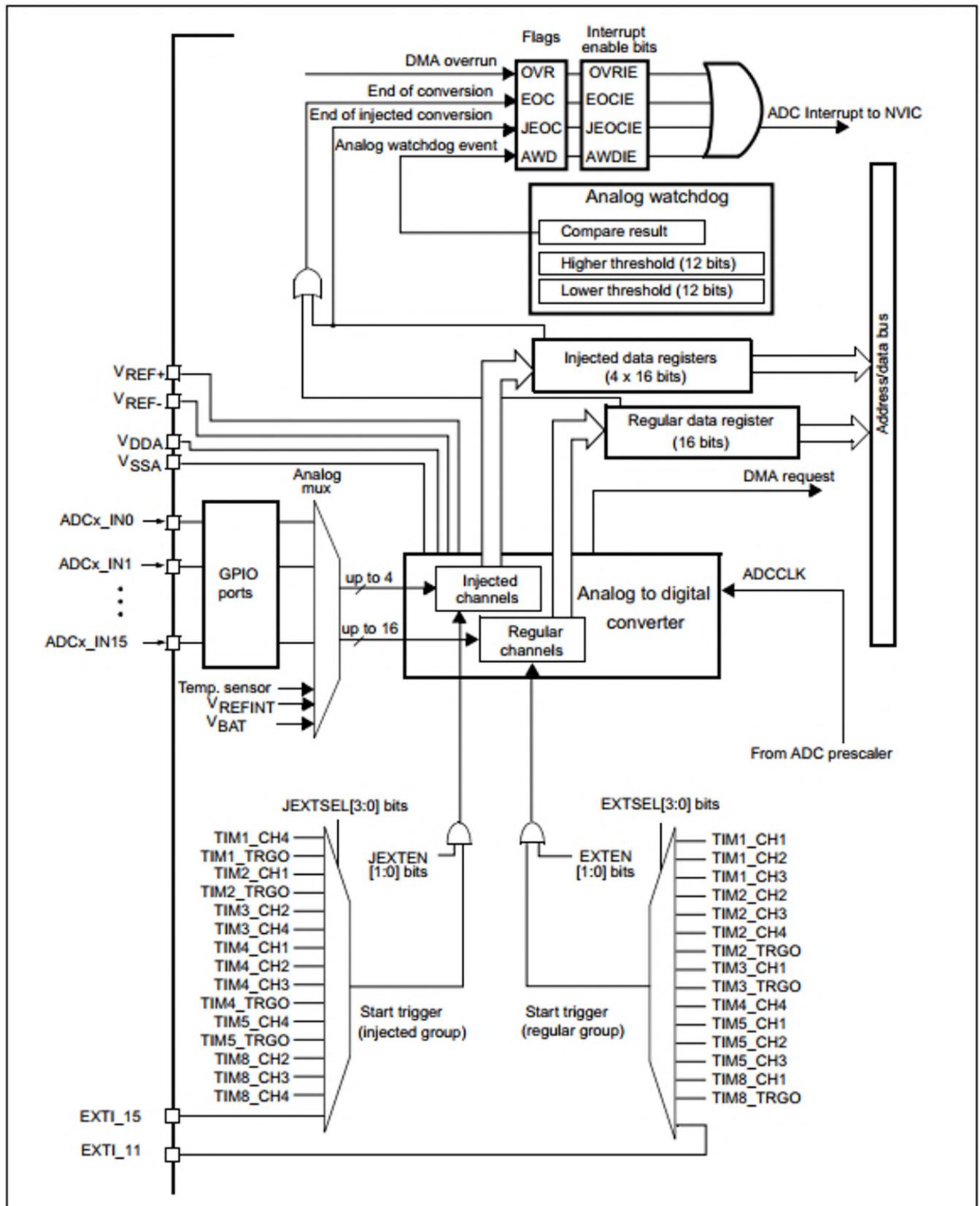
- Çevrim süresi hesabı için üç değere ihtiyaç var. Bunlar Cycles, Sampling Time ve Clock'tur.
- Cycles değeri seçilen Resolution değerine bağlıdır.
- Sampling Time ve Clock değerleri ise istediğimiz çevrim süresine göre değiştirebiliriz.
- Clock değeri ADC'nin bağlı olduğu clock hattına bağlıdır.
- Tüm işlemcilerde aynı mantıktır fakat formül işlemciye göre farklılık gösterebilir bunun için kaynaklardan bakılması gereklidir.

$$T_{\text{conv}} = \frac{\text{Sampling time} + \text{Cycles}}{\text{ADC CLOCK}}$$

## Çalışma Modları

- **Single Conversion Mode** (Tek Dönüşüm Modu): Bu mod, bir tek dönüşüm gerçekleştirildikten sonra ADC'nin otomatik olarak durmasını sağlar. Her dönüşüm, başlatma komutu ile başlatılır ve tamamlandığında ADC otomatik olarak durur.
- **Continuous Conversion Mode** (Sürekli Dönüşüm Modu): Bu modda ADC, başlatıldığı andan itibaren sürekli olarak dönüşümler gerçekleştirir. Otomatik durma olmadığı için dönüşümler devam eder, kullanıcı tarafından durdurulana kadar devam eder.
- **Scan Mode** (Tarama Modu): Bu modda ADC, belirli bir kanal listesini otomatik olarak tarama yeteneğine sahiptir. Tarama modu, birden fazla kanalı tek bir dönüşüm başlatma komutu ile sırayla ölçmeyi sağlar.
- **Discontinuous Mode** (Kesikli/Süreksiz Mod), kullanıcı belirli bir kanal listesinin ardışık olarak ölçülmesini sağlayabilir. Ancak, kanal arasında belirli bir gecikme bulunabilir.

## Birim Yapısı



## 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
0x00	ADC_SR	Reserved																															
	Reset value																																
0x04	ADC_CR1	Reserved				OVRIE	RES[1:0]		AWDEN	JAWDEN	Reserved				DISC NUM [2:0]			JDISCEN	DISCEN	JAUTO	AWD SGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]							
	Reset value						0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x08	ADC_CR2	Re se rv ed	SWSTART	EXTEN[1:0]	EXTSEL [3:0]			Re se rv ed	JSWSTART	JEXTEN[1:0]	JEXTSEL [3:0]			Reserved				ALIGN	EOCS	DDS	DMA	Reserved				CONT	ADON						
	Reset value		0	0	0	0	0	0	0	0	0	0	0					0	0	0	0					0	0						
0x0C	ADC_SMPR1	Sample time bits SMPx_x																															
	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	0	
0x10	ADC_SMPR2	Sample time bits SMPx_x																															
	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	0	
0x14	ADC_JOFR1	JOFFSET1[11:0]																															
	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	0	
0x18	ADC_JOFR2	JOFFSET2[11:0]																															
	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	0	
0x1C	ADC_JOFR3	JOFFSET3[11:0]																															
	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	0	
0x20	ADC_JOFR4	JOFFSET4[11:0]																															
	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	0	
0x24	ADC_HTR	HT[11:0]																															
	Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0x28	ADC_LTR	LT[11:0]																															
	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	0	
0x2C	ADC_SQR1	Reserved	L[3:0]				Regular channel sequence SQx_x bits																										
	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	
0x30	ADC_SQR2	Reserved	Regular channel sequence SQx_x bits																														
	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	
0x34	ADC_SQR3	Reserved	Regular channel sequence SQx_x bits																														
	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	
0x38	ADC_JSQR	Reserved	JL[1:0]				Injected channel sequence JSQx_x bits																										
	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	
0x3C	ADC_JDR1	JDATA[15:0]																															
	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	0	
0x40	ADC_JDR2	JDATA[15:0]																															
	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	0	
0x44	ADC_JDR3	JDATA[15:0]																															
	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	0	
0x48	ADC_JDR4	JDATA[15:0]																															
	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	0	
0x4C	ADC_DR	Regular DATA[15:0]																															
	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	0	0

- **ADC\_SR (Status Register)**, ADC durumunu izleyen bu register, dönüşüm tamamlandığında, taşıma veya analog bekçi olaylarının gerçekleştiğini belirten bayrakları içerir.
- **ADC\_CR1 (Control Register 1)**, Bu register, dönüşüm kesmelerini etkinleştirme, scan modunu kontrol etme, discontinuous modu ve enjekte dönüşümleri yönetme gibi temel ADC kontrol ayarlarını içerir.
- **ADC\_CR2 (Control Register 2)**, ADC'nin genel kontrolünü sağlayan bu register, ADC'nin etkinleştirilmesi, continuous conversion modu, DMA modu, kalibrasyon ve harici tetikleme seçenekleri gibi ayarları içerir.
- **ADC\_SMPR1 ve ADC\_SMPR2 (Sampling Time Register 1 ve 2)**: Örnekleme süresini belirleyen bu registerler, her bir kanalın örnekleme süresini ayarlamınızı sağlar.
- **ADC\_DR (Data Register)**, Dönüşüm sonuçlarını depolar; yani ADC tarafından ölçülen analog sinyalin dijital karşılığını içerir.

- ADC'deki "common registerlar," birden fazla ADC modülünün ortak kullanıldığı durumlar için genel ayarları ve durumu izlemek için tasarlanmış registerlardır. Bu registerlar, birden fazla ADC'nin ortak özelliklerini kontrol etmek ve izlemek için kullanılır.
  - **ADC\_CSR (Common Status Register):** ADC modülünün genel durumunu gösteren bu register, özellikle birden fazla ADC'nin kullanıldığı durumlarda ortak durumu izlemek için kullanılır.
  - **ADC\_CCR (Common Control Register):** Bu register, ortak ayarları içerir. Örneğin, referans voltajlarını (VREF+ ve VREF-) belirlemek gibi genel ADC kontrol parametrelerini içerir.
  - **ADC\_CDR (Common Data Register):** Birden fazla ADC kullanıldığında, çeşitli ADC'lerden gelen verileri depolar.

## Ölçüm Yöntemleri

- ADC ölçümlerini almak için kullanılan farklı yöntemler şunlardır: Polling, Interrupt ve DMA
  - <http://www.elektrobot.net/stm32-adc-kullanimi-polling-interrupt-ve-dma/> ile  
<https://controllerstech.com/stm32-adc-single-channel/> linkten Polling, Interrup ve DMA metodu kullanarak yapılan örnekleri inceleyebiliriz.
  - Polling** yöntemi, mikrodenetleyici ADC'nin çevrim süresince farklı bir işlem yapmaz ve çevrimin bitmesini bekler. Yapılacak ölçümün çok hızlı olmasının gerekmemiği yada uzun zaman aralıklarında tek ölçüm yapılmasının yeterli olduğu durumlarda sıklıkla kullanılır.
  - Interrupt** yöntemi, ADC dönüşümü tamamlandığında bir kesme çağrıları gerçekleşir. Böylece mikrodenetleyicinin başka işlerle meşgulken dahi ADC verilerini işlemesine izin verir. Daha karmaşık uygulamalarda, dönüşüm tamamlandığında hemen yanıt verilmesi gereken durumlar için uygundur. Verimli kullanım, mikrodenetleyicinin diğer görevlere odaklanması sağlar.
  - DMA** yöntemi, ADC sonuçları doğrudan belleğe kopyalanır, bu da CPU'nun dahil olmadan çalışmasına olanak tanır. Büyük veri setlerini hızlı bir şekilde işlemek ve mikrodenetleyicinin CPU'sunu diğer görevlere odaklamak için uygundur. Bellek yönetimi konusunda dikkatlice ele alınması gerekebilir.
  - DMA'nın Interrupt ile kullanımından en büyük farkı, ADC' nin çevrimi tamamladıktan sonra elde ettiği değeri hafıza bölgесine DMA tarafından yazılmıştır. Böylece mikrodenetleyici hiç bir şekilde ADC işlemleri ile meşgul olmaz. Özellikle çok sayıda ölçümün ard arda ve hızlı yapılmasının istendiği durumlarda DMA kullanılır.

# 04 DAC

5 Mayıs 2021 Çarşamba 08:02

## 04 DAC

### Giriş

- DAC, "Digital-to-Analog Converter" dijital sinyalleri analog sinyallere dönüştürmek için kullanılır. Genellikle mikrodenetleyiciler, bilgisayarlar, ses sistemleri ve diğer dijital cihazlar gibi dijital veri kaynaklarından gelen dijital verileri, analog çıkış cihazlarına (örneğin hoparlörler veya ses sistemleri) uygun bir şekilde aktarmak için kullanılırlar.
- STM32F407, 0-3.3 V arasında tüm gerilimleri çıkış olarak vermemizi sağlar.
- STM32F407 mikrodenetleyicisi içerisinde dahili olarak 12 bit tampona sahip, iki adet DAC birimi bulunur. Bu birimler sayesinde dijital bir veriyi analog bir veriye dönüştürerek çıkış üretilebilir.
- STM32F407'ye ait DAC birimleri 8 bit veya 12 bit değerinde çıkış üretilebilirler.
- 12 bit değerinde kullanılırken, veri 16 bitlik kaydedici içerisinde sola veya sağa dayalı şekilde kullanılabilir.
- DAC biriminin önemli özelliklerinden bir tanesi, gürültü veya sinyali üretebilme özelliğidir.
- Üçgen dalga üretebilme özelliğine sahiptir.
- DAC birimleri APB1 veri yoluna bağlıdır, kullanmak için aktif etmek gereklidir.
- DAC için hangi pin/pinler kullanılacaksa ilgili pin/pinler GPIOA->CRL registerinden analog moda alınmalıdır.

### Çözünürlük

- STM32'de DAC çözünürlüğünü artırmak için Vref+ girişi bulunmaktadır fakat bu pin yüksek işlemcilerde bulunmaktadır. Vref+ ve Vref- pini bulunmayan işlemcilerde bu pinler dahili olarak **VDDA** ve **VSSA**'ya bağlıdır. VDDA ve VSSA ise VDD ile VSS'ye bağlanması zorunludur. Buradanda Vref+ geriliminin besleme gerilimini geçemeyeceğini anlıyoruz.

$$\text{DACoutput} = V_{\text{REF}} \times \frac{\text{DOR}}{4096}$$

- Yukarıdaki ifade ile DAC çıkış voltajı hesaplanır. Biz DAC değerlerimizi DHR registerına yazarız ve tetikleme sonucunda DHR'deki veri DOR registerına aktarılır, DOR registerını sadece okuyabiliriz.
- 12 bitlik çözünürlüğe sahip bir DAC biriminin referans gerilimleri Vss = 0 V, Vdd = +3 V ele alınır ise, adım başına üreteceği voltaj şu şekilde hesaplanır;  $\text{DACoutput} = V_{\text{REF}}/4095$   
Buradan adım başına düşen voltaj,  $\text{DACoutput}=3V/4095 = 732,600732$   
Örneğin 1V elde etmek isteniyorsa:  $1/0,000732600 = 1365$  değeri elde edilir.

### Çalışma Modları

- STM32 mikrodenetleyicilerinde DAC modülü genellikle tek kanal, çift kanal, üçgen dalga ve gürültü oluşturma modları gibi farklı çalışma modlarına sahiptir.
- **Tek Kanal Modu**, Tek bir DAC kanalı üzerinden analog çıkış sağlar. Örneğin, STM32 mikrodenetleyicilerinde "DAC\_Channel\_1" kullanarak tek kanal modunda DAC'ı kullanabilirsiniz.
- **Çift Kanal Modu**, iki DAC kanalı üzerinden bağımsız olarak analog çıkış sağlar. Örneğin, STM32 mikrodenetleyicilerinde "DAC\_Channel\_1" ve "DAC\_Channel\_2" kullanarak çift kanal modunda DAC'ı kullanabilirsiniz.
- **Üçgen Dalga Modu**, DAC, üçgen dalga formunu üretebilir. Bu modda, DAC çıkışı belirli bir frekansta bir üçgen dalga formunu takip eder.
- **Gürültü Oluşturma Modu**, DAC, belirli bir frekansta gürültü sinyali üretebilir. Bu modda, DAC çıkışı belirli bir frekansta rasgele değerler üreterek bir gürültü sinyali oluşturur.

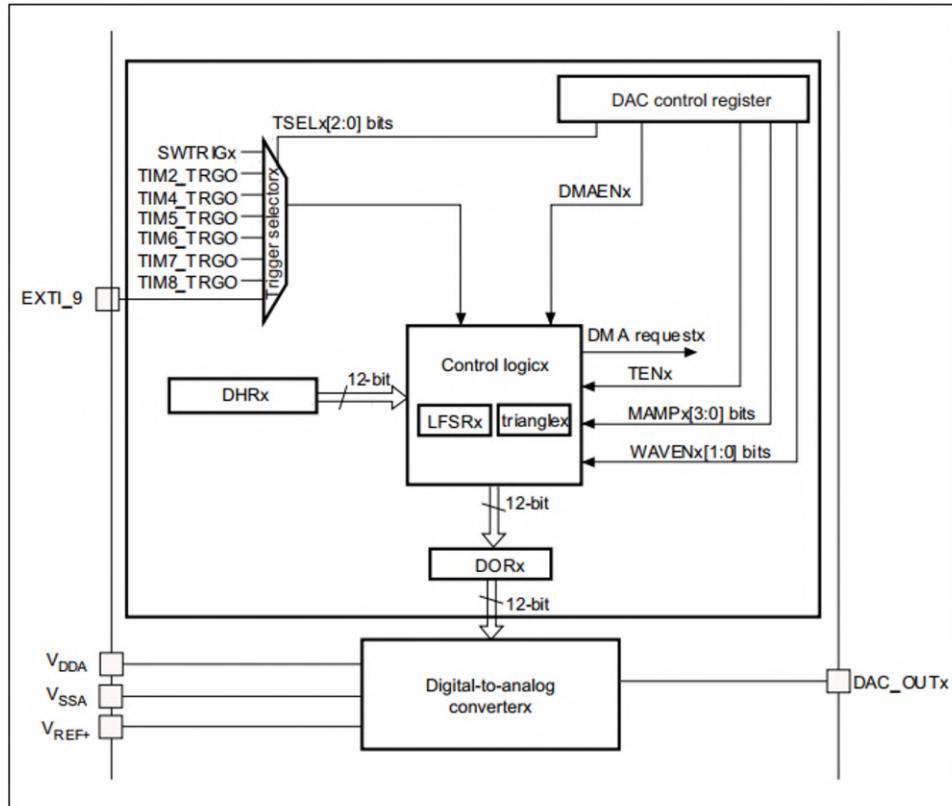
### Tetikleme İşlemleri

- Genellikle yazılımsal ve harici tetikleme (triggering) yöntemleri ile kullanılabilir. Bu yöntemler, DAC'nın çıkışını kontrol etmek ve çıkış verisini belirli bir zamanlama veya olaya bağlamak için kullanılır.
- **Software Triggering**, Yazılımsal tetikleme, mikrodenetleyici yazılımı tarafından kontrol edilen bir tetikleme yöntemidir. Yazılım, DAC çıkışını başlatmak veya durdurmak için özel bir komut kullanır. Bu yöntem, zamanlama ile ilgili hassas kontrol gerektiren durumlarda kullanışlıdır. Örneğin, bir zamanlayıcı kesmesi veya belirli bir durum gerçekleştiğinde DAC çıkışını güncellemek için yazılımsal tetikleme kullanılabilir.
- **External Triggering**, DAC modülünü dış bir olaya (örneğin, bir zamanlayıcı kesmesi, bir GPIO değişikliği veya başka bir harici sinyal) bağlamak anlamına gelir. Harici bir sinyal algılandığında veya belirli bir durum gerçekleştiğinde, DAC çıkışını güncellemek için harici bir sinyal kullanılabilir.

## Farklılıklar

- DAC ve PWM, her ikisi de dijital sinyalleri analog sinyallere dönüştürmek için kullanılan yöntemlerdir, ancak farklı çalışma prensiplerine sahiptirler.
- DAC, doğrudan dijital değerleri analog voltaj veya akıma dönüştürken, PWM, darbe genişliği modülasyonu yoluyla bir analog etki oluşturur.
- DAC, genellikle doğrudan analog çıkış sağlar ve daha hassas bir çözünürlük sunabilir. PWM ise daha çok göreceli ve yaklaşık bir çözünürlük sağlar.
- DAC, genellikle özel bir entegre devre içerirken, PWM, genellikle bir mikrodenetleyici tarafından kontrol edilir.
- DAC, yüksek hassasiyet gerektiren ses uygulamalarında daha tercih edilebilirken, PWM, motor hız kontrolü gibi uygulamalarda daha uygun olabilir.

## Birim Yapısı



## Register

Offset	Register	31	30	29	DMAUDRIE2	28	DMAEN2	27	26	25	24	23	22	21	20	19	18	TEN2	BOFF2	17	EN2	16	15	14	13	DMAUDRIE1	12	DMAEN1	11	10	9	8	7	6	5	4	3	2	1	0			
0x00	DAC_CR	Reserved			MAMP2[3:0]	WAVE2[2:0]	TSEL2[2:0]								Reserved																												
0x04	DAC_SWTRIGR																																										
0x08	DAC_DHR12R1																																										
0x0C	DAC_DHR12L1																																										
0x10	DAC_DHR8R1																																										
0x14	DAC_DHR12R2																																										
0x18	DAC_DHR12L2																																										
0x1C	DAC_DHR8R2																																										
0x20	DAC_DHR12RD	Reserved																																									
0x24	DAC_DHR12LD																																										
0x28	DAC_DHR8RD																																										
0x2C	DAC_DOR1																																										
0x30	DAC_DOR2																																										
0x34	DAC_SR	Reserved	DMAUDR2																																								

- **DAC\_CR (Control Register)**, DAC'nin genel kontrolünü sağlayan bu register, örneğin çıkış voltaj seviyesi, çıkış güçlendirme ve trigger seçeneklerini içerir.
- **DAC\_SWTRIGR (Software Trigger Register)**, yazılım tetikleme işlemlerini kontrol etmek için kullanılır.
- **DAC\_DHR (Data Holding Register)**, bu register'lar, DAC'ye gönderilecek dijital veriyi içerir.
- **DAC\_SR (Status Register)**, DAC durumunu izlemek için kullanılır.
- **DAC\_DOR (Data Output Register)**, DAC'nın çıkışından okunan gerçek zamanlı dijital çıkış verisini temsil eder. Dönüştürülen analog sinyalin temsil ettiği dijital değeri içerir.

# 05 DMA

5 Mayıs 2021 Çarşamba 08:03

## 05 DMA

### Giriş

- <https://mikrodunya.wordpress.com/2016/06/23/dma-direct-memory-access-dogrudan-bellek-erisimi/>
- DMA(Direct Memory Adres) gelişmiş mikrodenetleyicilerde peripheral-memory veya memory-memory arasındaki veri transferlerini hiç bir CPU işlemini kullanmadan sağlaması amacıyla oluşturulmuştur.
- Çok veri alışverişi yapıldığı durumlarda kullanılması gereklidir.
- Çeşitli çevre birimlerinden okuduğumuz verileri bir değişkenle atarız. Bu değişkenler RAM'de depolanır. Bu işlem normalde çevre birimlerinde okunan verinin CPU'ya alınıp ardından RAM'e yazılır. Ancak CPU kullanımı hem işlemciyi yorar hemde kayıplara yol açar.
- DMA sayesinde verileri direkt olarak **RAM'e** yazma imkanı buluruz.
- DMA donanımı CPU'dan bağımsız olarak verilerimizi peripheral'dan hafızaya, hafızadan peripheral'a ve hafızadan hafızaya olmak üzere hızlı bir şekilde kaynak adresinden hedef adresine aktarır.

**peripheral -> memory**

**memory -> peripheral**

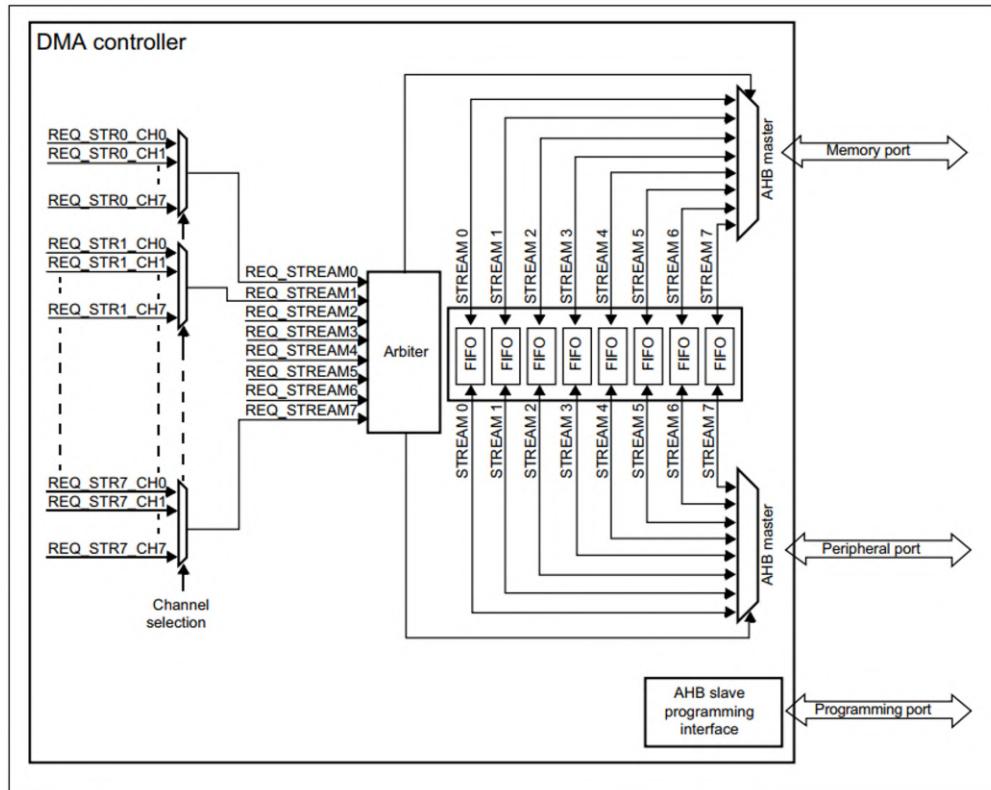
**memory -> memory**

- Bu sayede CPU'nun yükünü hafifletmiş oluruz. Sistem sanki 2 CPU ile çalışmış gibi düşünebiliriz. Örneğin bilgisayarlarında bulunan 4 gerçek 4 sanal çekirdekteki sanal, aslında DMA diyebiliriz. DMA isteği için çevresel birim tarafından (ADC, DAC, I2C vs) DMA kontrolcüsüne istek gönderilir, kontrolcüde bu isteğin sırası gelince ilgili çevresel birime geri bildirimde bulunur ve işlem kaynak adresinden hedef adresine doğru gerçekleşir.
- STM32F4'te iki adet DMA vardır. DMA1'in DMA 2'den kanal 1'in kanal 2'den yüksek olduğu bilinmektedir. Öncelik sırası belirtmek için dört seviye vardır. Low, Medium, High, Very High.

Aynı anda birçok kanal kullanıldığında hangi kanalın öncelik değeri fazla ise ilk o kanal alınır.

DMA'lar paralel olarak çalışmazlar, seri olarak çalışırlar. Bu nedenle hangisinin sırası geldi ise o anda o çalışır.

### Birim Yapısı



### Register

Offset	Register	31	30	29	28
0x0000	<b>DMA_LISR</b>	Reserved			
0x0004	<b>DMA_HISR</b>	Reserved			
0x0008	<b>DMA_LIFCR</b>	Reserved			
0x000C	<b>DMA_HIFCR</b>	Reserved			
0x0010	<b>DMA_S0CR</b>	Reserved			
0x0014	<b>DMA_S0NDTR</b>	Reserved			
0x0018	<b>DMA_S0PAR</b>	PA[31:0]			
		Reset value			
0x001C	<b>DMA_S0M0AR</b>	M0A[31:0]			
		Reset value			
0x0020	<b>DMA_S0M1AR</b>	M1A[31:0]			
		Reset value			
0x0024	<b>DMA_S0FCR</b>	Reserved			
		Reset value			

- **DMA\_LISR ve DMA\_HISR (Low/High Interrupt Status Register)**, DMA'nın düşük ve yüksek öncelikli kesmelerin durumunu izleyen register'lardır. Her bir bit, ilgili DMA kanalındaki bir kesmeyi temsil eder.
  - **DMA\_LIFCR ve DMA\_HIFCR (Low/High Interrupt Flag Clear Register)**, DMA'nın düşük ve yüksek öncelikli kesme bayraklarını temizlemek için kullanılır. Her bir bit, ilgili DMA kanalındaki bir kesme bayrağını temsil eder.
  - **DMA\_SxCR (Stream x Configuration Register)**, DMA'nın belirli bir kanalının yapılandırma register'ıdır. Kanalın çalışma modu, transfer yönü, veri genişliği, bellek ve perifer adresi inkrement modu gibi özellikleri içerir.
  - **DMA\_SxNDTR (Stream x Number of Data Register)**, ilgili DMA kanalında aktarılacak veri miktarını belirten register'dır.
  - **DMA\_SxPAR (Stream x Peripheral Address Register)**, DMA'nın belirli bir kanalındaki perifer başlangıç adresini belirten register'dır.
  - **DMA\_SxMxAR ve DMA\_SxMxAR (Stream x Memory 0/1 Address Register)**, DMA'nın belirli bir kanalındaki bellek başlangıç adreslerini belirten register'lardır. Bazı STM32 modellerinde birden fazla bellek adresi kullanılabilir.
  - **DMA\_SxFCR (Stream x FIFO Control Register)**, DMA FIFO (First In, First Out) kontrolünü sağlayan register'dır. FIFO'nun kullanılması, DMA transfer performansını artırabilir.

# 06 TIMER

5 Mayıs 2021 Çarşamba 08:02

## 06 TIMER

### Giriş

- Timer modülünün temel görevi zamanlama yapmaktadır. İşlemci frekanasına bağlı olarak çalışırlar. Dışarıdan gelen pulse darbelerini sayarlar. İşlemciye tanıtılan bir süre ile, geçen süreyi karşılaştırma ve belli bir süre sonunda kesme üretme gibi işlemlerde kullanılırlar.
  - Sayıcı birimi sabit bir frekans kaynağı ile besleniyorsa Timer olarak çalışır. Zamanlayıcının bir adımı 1/f süresine denk gelir. Örneğin 1 kHz ile beslenen bir zamanlayıcının her adımı 1 ms demektir.
  - 1kHz ile beslenen zamanlayıcıyı t1 anında okuduğumuzda değeri 100, t2 anında okuduğumuzda değeri 250 ise, t2-t1 arasında geçen süre 150ms demektir. Zamanlayıcılar ile bu şekilde zaman ölçümü ya da periyodik işlemlerin gerçekleştirilebilmesini sağlarlar.
  - Timer, belirli bir süre veya sayıı gerçekleştirdikten sonra, sayaç değeri belirli bir sınırı aşarsa veya taşarsa, overflow durumu ortaya çıkar. Bu zamanlayıcı bir belirli sayıya kadar sayıyorsa sayaç bu sayıya ulaştığında, taşıma **overflow** gerçekleşir ve sayaç sıfırlanarak yeniden başlar.
  - **Capture**, zamanlayıcının mevcut değerini özel bir kaydediciye kopyalama işlemidir. Bu, bir dış olayın gerçekleştiği belirli bir zamanı yakalamak için kullanılabilir. Örneğin, dışardan gelen sinyalin belirli bir durumu algandığında, zamanlayıcı değeri bu anda "yakalanır" ve kaydedilir. Bu, belirli olayların zaman damgalarını elde etmek için sıkılıkla kullanılır.
  - **Compare**, zamanlayıcı değerini bir belirli değerle karşılaştırma işlemini ifade eder. Zamanlayıcı, belirli bir değere ulaştığında veya onu geçtiğinde, bu bir olayın tetiklenmesine neden olabilir. Örneğin, belirli bir zaman geçtikten sonra bir işlemi başlatmak için compare özelliğini kullanılabilir. Bu, periyodik işlemleri kontrol etmek veya belirli bir süreyi takip etmek için yaygın olarak kullanılır.
  - **Pulse Width Modulation (PWM)**, genellikle bir dijital sinyalin darbe genişliğini modüle etme tekniğini ifade eder. PWM, bir sinyalin belirli bir süre boyunca HIGH ve belirli bir süre boyunca LOW olduğu bir sinyal üretir. Bu modülasyon tekniği, analog sinyal davranışını taklit etmek veya kontrol etmek için yaygın olarak kullanılır.
- Çoğu mikrodenetleyicide PWM birimleri de Timer unitelerine bağlı olarak çalışırlar.
- STM32F407VG işlemcisinde toplam 17 adet timer birimi bulunur.

10 adet **General Purpose**, 2 adet **Advanced Control**, 2 adet **Basic**, 1 adet **Independent Watchdog (IWDG)**, 1 adet **Window Watchdog (WWDG)** timer, 1 adet **Systemtick** timer var.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)
Advanced -control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	168
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	168
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	168
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	42	84
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	42	84
Basic	TIM6,	16-bit	Up	Any integer between 1 and 65536	N/A	0	N/A	42	84

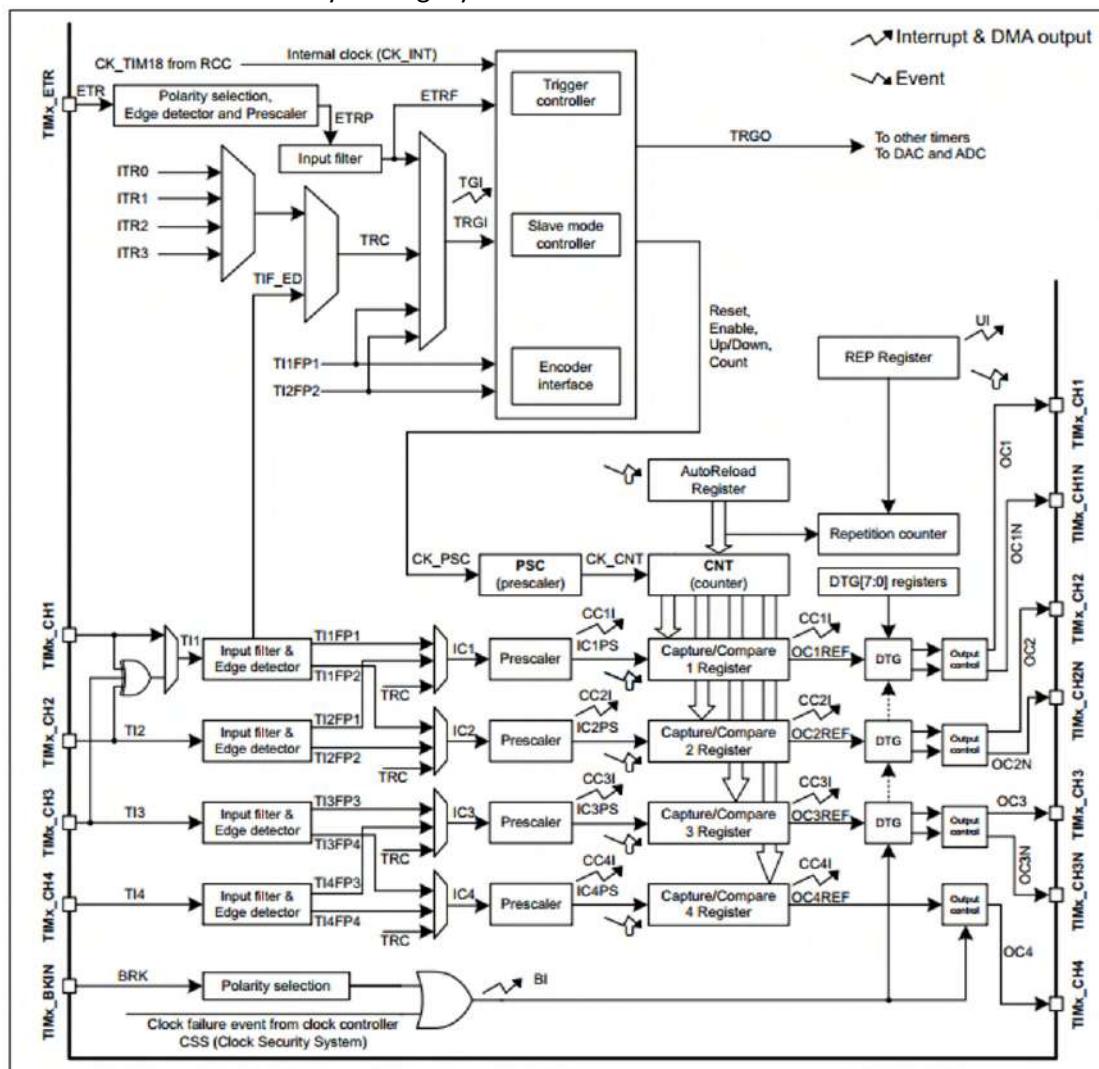
	TIM14	16-bit	Up	between 1 and 65536	No	1	No	42	84
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	42	84

## Advanced Control

TIM1, TIM8

- TIM1 ve TIM8, yüksek hızlı APB2 veri yolu (84 MHz) üzerinde bulunurlar. Eğer APB2 prescaleri değişkeni 1 değerinden farklı ise bu timer birimlerinin saat frekansı, APB2'nin frekans değerinin iki katı olur. Yani, bu timer birimlerinin maksimum çalışma frekansları 168 MHz olabilir.
  - TIM1 ve TIM 8 birimleri 16 bitlik sayıciya sahiptirler.
  - Bu sayıcılar; yukarı, aşağı ve merkezlenmiş modlarda sayma yapabilirler.
  - Bu sayıcların otomatik geri yükleme özellikleri bulunmaktadır.
  - Bu timer birimlerinde 4x16 adet yüksek çözünürlüklü capture/compare kapalı da bulunur.

Bu kanallar giriş çıkış olarak ayarlanabilir, çıkış karşılaştırabilir, PWM sinyali üretebilir, sinyal yakalayabilir ve harici bir PWM sinyalini algılayabilirler.



0x08	TIMx_SMCR	Reserved		ETF [1:0]	TS[2:0]	SMS[2:0]	0	ETF [1:0]	TS[2:0]	SMS[2:0]	
							0	ETF [1:0]	TS[2:0]	SMS[2:0]	
0x0C	TIMx_DIER	Reserved		TDE	CC4DE	CC4DE	0	BIE	0	UIE	
				0	0	0	0	0	0	0	
0x10	TIMx_SR	Reserved		CC4OF	CC3OF	CC3OF	0	CC4IF	0	CC4IE	
				0	0	0	0	0	0	0	
0x14	TIMx_EGR	Reserved		CC4OF	CC3OF	CC3OF	0	CC3IF	0	CC3IE	
				0	0	0	0	0	0	0	
0x18	TIMx_CCMR1 Output compare mode	Reserved		OC2CE	OC2M [2:0]	OC2PE	CC2S [1:0]	OC1CE	OC1M [2:0]	CC1S [1:0]	
				0	0	0	0	0	0	0	
0x18	TIMx_CCMR1 Input capture mode	Reserved		IC2F[3:0]	IC2PSC [1:0]	CC2S [1:0]	IC1F[3:0]	IC1PSC [1:0]	CC1S [1:0]	CC1S [1:0]	
				0	0	0	0	0	0	0	
0x1C	TIMx_CCMR2 Output compare mode	Reserved		OC4CE	OC4M [2:0]	OC4PE	CC4S [1:0]	OC3CE	OC3M [2:0]	CC3S [1:0]	
				0	0	0	0	0	0	0	
0x1C	TIMx_CCMR2 Input capture mode	Reserved		IC4F[3:0]	IC4PSC [1:0]	CC4S [1:0]	IC3F[3:0]	IC3PSC [1:0]	CC3S [1:0]	CC3S [1:0]	
				0	0	0	0	0	0	0	
0x20	TIMx_CCER	Reserved		CC1NP Reserved	CC4P	CC4E	CC3E	CC2P	CC2E	CC1E	
				0	0	0	0	0	0	0	
0x24	TIMx_CNT	Reserved		CNT[15:0]							
				0	0	0	0	0	0	0	0
0x28	TIMx_PSC	Reserved		PSC[15:0]							
				0	0	0	0	0	0	0	0
0x2C	TIMx_ARR	Reserved		ARR[15:0]							
				1	1	1	1	1	1	1	1
0x30	TIMx_RCR	Reserved		REP[7:0]							
				0	0	0	0	0	0	0	0
0x34	TIMx_CCR1	Reserved		CCR1[15:0]							
				0	0	0	0	0	0	0	0
0x38	TIMx_CCR2	Reserved		CCR2[15:0]							
				0	0	0	0	0	0	0	0
0x3C	TIMx_CCR3	Reserved		CCR3[15:0]							
				0	0	0	0	0	0	0	0
0x40	TIMx_CCR4	Reserved		CCR4[15:0]							
				0	0	0	0	0	0	0	0
0x44	TIMx_BDTR	Reserved		MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK [1:0]	DT[7:0]
				0	0	0	0	0	0	0	0
0x48	TIMx_DCR	Reserved		DBL[4:0]							
				0	0	0	0	0	0	0	0
0x4C	TIMx_DMAR	DMAB[31:0]								DBA[4:0]	
		Reset value	0	0	0	0	0	0	0	0	0

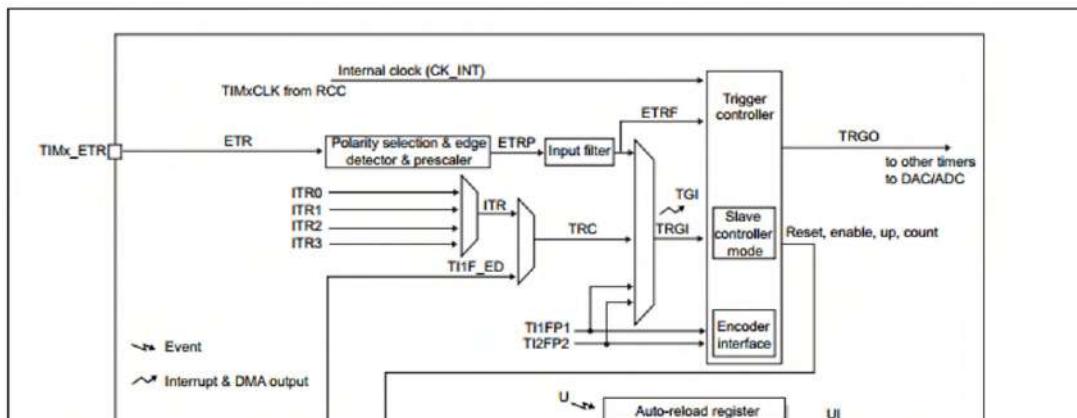
- **TIMx\_CR1 (Control Register 1)**, Timer'in genel kontrol ayarlarını içerir. Timer'ı etkinleştirme, zamanlama

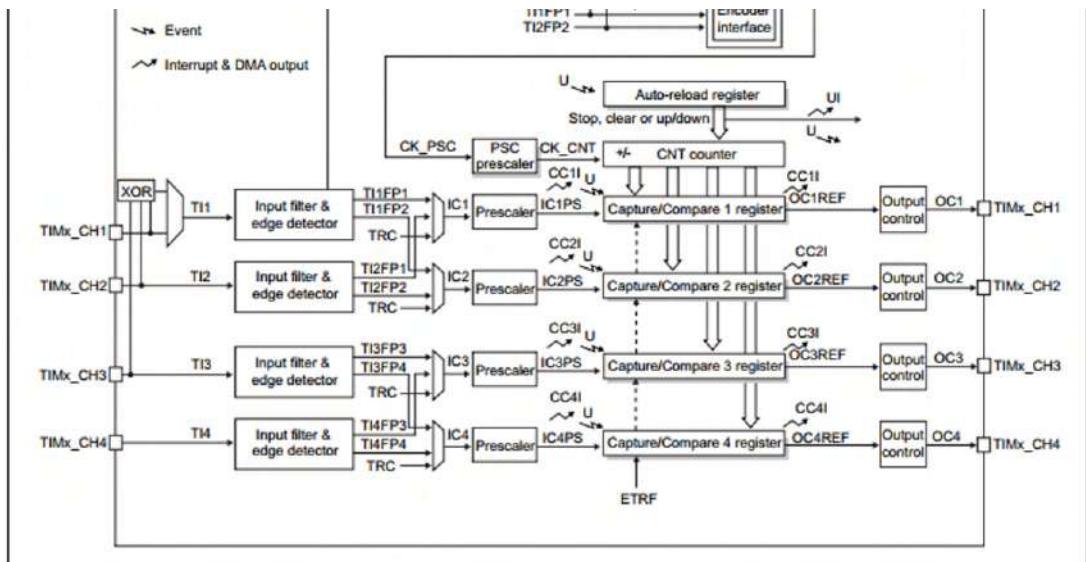
- **TIMx\_CR1 (Control Register 1)**, Timer'in genel kontrol ayarlarını içerir. Timer'ı etkinleştirme, zamanlama modu seçimi, otomatik yeniden başlatma etkinleştirme gibi ayarları içerir.
  - **TIMx\_CR2 (Control Register 2)**, Timer'in özel kontrol ayarlarını içerir. Bu register, master mode seçimi gibi özellikleri kontrol eder.
  - **TIMx\_SMCR (Slave Mode Control Register)**, Timer'in slave modunu kontrol eder. Dış bir kaynaktan senkronize olma veya bir başka timer'i takip etme gibi işlevleri içerir.
  - **TIMx\_DIER (DMA/Interrupt Enable Register)**, DMA ve kesme interrupt izinlerini kontrol eder. Belirli olayların tetiklenmesi durumunda bir kesme talebi veya DMA transferi başlatma gibi işlevleri etkinleştirir veya devre dışı bırakır.
  - **TIMx\_SR (Status Register)**, Timer'in durumuyla ilgili bilgileri içerir. Taşma, karşılaşılma olayları gibi çeşitli olayları takip eder.
  - **TIMx\_EGR (Event Generation Register)**, Olayların elle tetiklenmesini sağlar. Bu register üzerinden bir olay (event) hemen tetikleyebilirsiniz.
  - **TIMx\_CCMR1 ve TIMx\_CCMR2 (Capture/Compare Mode Register 1 ve 2)**, Capture/compare modu için ayarları içerir. Timer'in çeşitli modlarını, giriş ve çıkış ayarlarını belirler.
  - **TIMx\_CCER (Capture/Compare Enable Register)**, Capture/compare kanallarını etkinleştirme veya devre dışı bırakma işlemlerini kontrol eder.
  - **TIMx\_CNT (Counter Register)**, Timer'in ana sayaç değerini içerir. Bu register, zamanlayıcının sayma işlemini temsil eder.
  - **TIMx\_PSC (Prescaler Register)**, Timer'in ön bölücü prescaler değerini içerir. Bu değer, timer'in sayma hızını kontrol eder.
  - **TIMx\_ARR (Auto-Reload Register)**, Timer'in otomatik yeniden başlatma değerini içerir. Bu değer, sayacın bir döngü tamamlandığında otomatik olarak tekrar başlamasını sağlar.
  - **TIMx\_RCR (Repetition Counter Register)**, İleri dönüş (overflow) olayının tekrar sayısını kontrol eder.
  - **TIMx\_CCR1, TIMx\_CCR2, TIMx\_CCR3, TIMx\_CCR4 (Capture/Compare Register 1, 2, 3, ve 4)**, Capture/compare modunda kullanılan karşılaşılma değerlerini içerir. Bu değerler, belirli bir zaman noktasında veya karşılaşılma olayında kullanılır.
  - **TIMx\_BDTR (Break and Dead-Time Register)**, Timer'in kesme ve ölü zaman ayarlarını içerir.
  - **TIMx\_DCR (DMA Control Register)**, DMA transferlerini kontrol eder.
  - **TIMx\_DMAR (DMA Address Register)**, DMA transferleri için adres bilgisini içerir.

## General Purpose

TIM2, TIM3, TIM4, ve TIM5

- TIM2, TIM3, TIM4, ve TIM5 birimleri, düşük hızlı APB1 (42 MHz) veri yolu üzerinde bulunmaktadır. Eğer APB1 prescaler değeri 1 den farklı ise bu timerların clock frekansları beslendikleri frekansların 2 katına çıkar. Yani 84 MHz clock frekansına sahip olur.
  - TIM3 ve TIM4 16-bit'lik sayıcıya, TIM2 ve TIM5 32-bit'lik sayıcıya sahiptirler.
  - Bu sayıcılar up, down ve auto-reload modlarda sayma yapabilirler.
  - Ayrıca bu sayıcıların otomatik yükleme özellikleri de vardır.
  - 16-bit genişliğinde kontrol edilebilir prescaler değeri vardır.
  - Bu timer biriminde 4x16 adet yüksek çözünürlüktü capture/compare kanalı bulunur. Bu kanallar; Input Capture, Output Compare, PWM, One-Pulse'dır.
  - Dahili diğer Timer birimleri ile senkronizasyon
  - Interrupt ve DMA üretimi mevcuttur.
  - Clock kaynağı seçimi





0x28	TIMx_PSC	Reserved	PSC[15:0]											
			0	0	0	0	0	0	0	0	0	0	0	0
0x2C	TIMx_ARR	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)												ARR[15:0]
		Reset value	1	1	1	1	1	1	1	1	1	1	1	1
0x34	TIMx_CCR1	CCR1[31:16] (TIM2 and TIM5 only, reserved on the other timers)												CCR1[15:0]
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0
0x38	TIMx_CCR2	CCR2[31:16] (TIM2 and TIM5 only, reserved on the other timers)												CCR2[15:0]
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	TIMx_CCR3	CCR3[31:16] (TIM2 and TIM5 only, reserved on the other timers)												CCR3[15:0]
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0
0x40	TIMx_CCR4	CCR4[31:16] (TIM2 and TIM5 only, reserved on the other timers)												CCR4[15:0]
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0
0x48	TIMx_DCR	Reserved				DBL[4:0]				Reserved	DBA[4:0]			
						0	0	0	0		0	0	0	0
0x4C	TIMx_DMAR	Reserved				DMAB[15:0]								
						0	0	0	0	0	0	0	0	0
0x50	TIM2_OR	Reserved				Reserved	ITR1 RMP		Reserved					
							0	0						
0x50	TIM5_OR	Reserved				Reserved	IT4 RMP		Reserved					
							0	0						

- **TIMx\_CR1 (Control Register 1)**, Timer'in genel kontrol ayarlarını içerir. Timer'in etkinleştirme, zamanlama modu seçimi, otomatik yeniden başlatma etkinleştirme gibi ayarları içerir.
- **TIMx\_CR2 (Control Register 2)**, Timer'in özel kontrol ayarlarını içerir. Bu register, master mode seçimi gibi özelliklerini kontrol eder.
- **TIMx\_SMCR (Slave Mode Control Register)**, Timer'in slave (köle) modunu kontrol eder. Dış bir kaynaktan senkronize olma veya bir başka Timer'in takip etme gibi işlevleri içerir.
- **TIMx\_DIER (DMA/Interrupt Enable Register)**, DMA (Direct Memory Access) ve kesme (interrupt) izinlerini kontrol eder. Belirli olayların tetiklenmesi durumunda bir kesme talebi veya DMA transferi başlatma gibi işlevleri etkinleştirir veya devre dışı bırakır.
- **TIMx\_SR (Status Register)**, Timer'in durumuyla ilgili bilgileri içerir. Taşma, karşılaşırma olayları gibi çeşitli olayları takip eder.
- **TIMx\_EGR (Event Generation Register)**, Olayların elle tetiklenmesini sağlar. Bu register üzerinden bir olayı (event) hemen tetikleyebilirsiniz.
- **TIMx\_CCMR1 ve TIMx\_CCMR2 (Capture/Compare Mode Register 1 ve 2)**, Capture/compare modu için ayarları içerir. Timer'in çeşitli modlarını, giriş ve çıkış ayarlarını belirler.
- **TIMx\_CCER (Capture/Compare Enable Register)**, Capture/compare kanallarını etkinleştirme veya devre dışı bırakma işlemlerini kontrol eder.
- **TIMx\_CNT (Counter Register)**, Timer'in ana sayıç değerini içerir. Bu register, zamanlayıcının sayma işlemini temsil eder.
- **TIMx\_PSC (Prescaler Register)**, Timer'in ön bölücü (prescaler) değerini içerir. Bu değer, timer'in sayma hızını kontrol eder.
- **TIMx\_ARR (Auto-Reload Register)**, Timer'in otomatik yeniden başlatma değerini içerir. Bu değer, sayacın bir döngü tamamlandığında otomatik olarak tekrar başlamasını sağlar.
- **TIMx\_CCR1, TIMx\_CCR2, TIMx\_CCR3, TIMx\_CCR4 (Capture/Compare Register 1, 2, 3, ve 4)**, Capture/compare modunda kullanılan karşılaşırma değerlerini içerir. Bu değerler, belirli bir zaman noktasında veya karşılaşırma olayında kullanılır.
- **TIMx\_DCR (DMA Control Register)**, DMA transferlerini kontrol eder.
- **TIMx\_DMAR (DMA Address Register)**, DMA transferleri için adres bilgisini içerir.
- **TIMx\_OR (Option Register)**, Timer'in özel seçeneklerini kontrol eder. Bu register, özel özelliklerin etkinleştirilmesi veya devre dışı bırakılması için kullanılır.

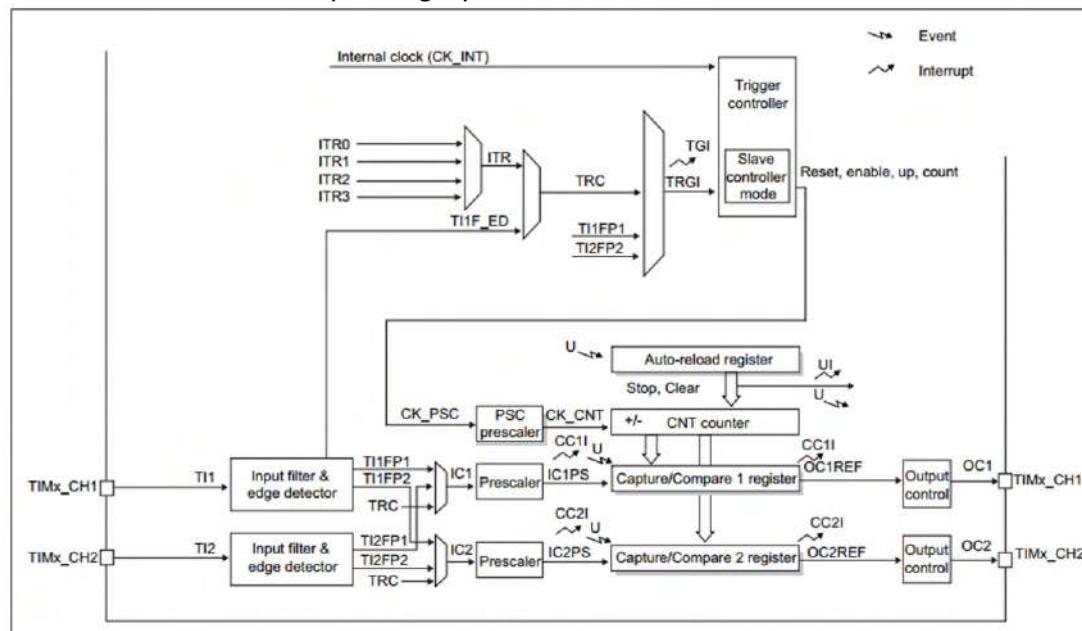
**TIM9, TIM10, TIM11, TIM12, TIM13, TIM14**

- TIM9 yüksek hızlı ADR2 (24 MHz) ve TIM12 düşük hızlı ADR1 (12 MHz) üzerinde bulunmaktadır.

etkinleştirilmesi veya devre dışı bırakılması için kullanılır.

#### TIM9, TIM10, TIM11, TIM12, TIM13, TIM14

- TIM9 yüksek hızlı APB2 (84 MHz) ve TIM12 düşük hızlı APB1 (42 MHz) üzerinde bulunmaktadır.
- Bu birimlerin frekansları diğerlerinde olduğu gibi veriyolu hızlarının iki katında çalışabilirler.
- TIM9 ve TIM12 birimleri 16 bitlik sayıcıya sahiptirler. Bu sayıçılara sadece yukarı sayma yapabilirler. Ayrıca bu sayıçılara otomatik geri yükleme özellikleri de bulunmaktadır.
- Bu timer birimlerinde 2x16 adet yüksek çözünürlüklü capture/compare kanalı da bulunmaktadır. Bu kanallar giriş öikiş olarak ayarlanabilir, çıkış karşılaştırabilir, PWM sinyali üretebilir, sinyal yakalayabilir ve harici bir PWM sinyalini algılayabilirler.
- TIM10 ve TIM11 yüksek hızlı APB2 (84 MHz) ve TIM13 ve TIM14 düşük hızlı APB1 (42 MHz) üzerinde bulunmaktadır. Bu birimlerin frekansları diğerlerinde olduğu gibi veriyolu hızlarının iki katında çalışabilirler.
- Bu birimler 16 bitlik sayıcıya sahiptirler. Bu sayıçılara sadece yukarı sayma yapabilirler. Ayrıca bu sayıçılara otomatik geri yükleme özellikleri de bulunmaktadır.
- Bu timer birimlerinde 2x16 adet yüksek çözünürlüklü capture/compare kanalı da bulunmaktadır. Bu kanallar giriş öikiş olarak ayarlanabilir, çıkış karşılaştırabilir, PWM sinyali üretebilir, sinyal yakalayabilir ve harici bir PWM sinyalini algılayabilirler.



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									
0x00	TIMx_CR1	Reserved												CKD [1:0]		ARPE		Reserve d		OPM1		URS		UDIS		CEN		0														
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0											
0x0C	TIMx_DIER	Reserved																									CC1IE		0													
	Reset value																										UIF		0													
0x10	TIMx_SR	Reserved												CCTOF		Reserved												0		0												
	Reset value																																									
0x14	TIMx_EGR	Reserved																									UG		0													
	Reset value																										0		0													
0x18	TIMx_CCMR1 Output compare mode	Reserved												OC1M [2:0]		OC1PE		OC1FE		CC1S [1:0]																						
	Reset value																																									
	TIMx_CCMR1 Input capture mode	Reserved												IC1F[3:0]		IC1PSC [1:0]		IC1REF		CC1S [1:0]																						
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
0x20	TIMx_CCER	Reserved																										CC1NP		0												
	Reset value																											CC1P		0												

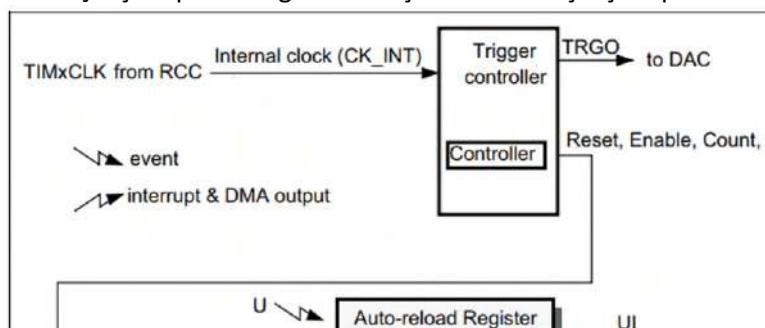
0x20	<b>TIMx_CCER</b>	Reserved			
		0	Reserved	CC1NP 0	CC1P 0
0x24	<b>TIMx_CNT</b>	Reserved	CNT[15:0]		
			0	0	0
0x28	<b>TIMx_PSC</b>	Reserved	PSC[15:0]		
			0	0	0
0x2C	<b>TIMx_ARR</b>	Reserved	ARR[15:0]		
			0	0	0
0x34	<b>TIMx_CCR1</b>	Reserved	CCR1[15:0]		
			0	0	0
0x50	<b>TIMx_OR</b>	Reserved			
			0	1	RMP 0

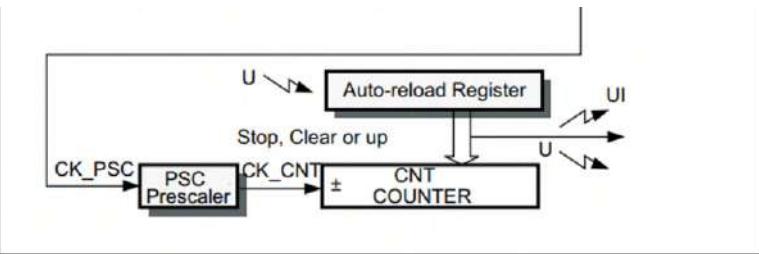
- **TIMx\_CR1 (Control Register 1)**, Timer'in genel kontrol ayarlarını içerir. Etkinleştirme, zamanlama modu seçimi, otomatik yeniden başlatma etkinleştirme ve diğer bazı genel ayarları içerir.
- **TIMx\_DIER (DMA/Interrupt Enable Register)**, DMA (Direct Memory Access) ve kesme (interrupt) izinlerini kontrol eder. Belirli olayların tetiklenmesi durumunda bir kesme talebi veya DMA transferi başlatma gibi işlevleri etkinleştirir veya devre dışı bırakır.
- **TIMx\_SR (Status Register)**, Timer'in durumuyla ilgili bilgileri içerir. Taşma, karşılaşıştırma olayları gibi çeşitli olayları takip eder.
- **TIMx\_EGR (Event Generation Register)**, Olayların elle tetiklenmesini sağlar. Bu register üzerinden bir olayı event hemen tetikleyebilirsiniz.
- **TIMx\_CCMR1 (Capture/Compare Mode Register 1)**, Yakalama/karşılaştırma modu için ayarları içerir. Timer'in çeşitli modlarını, giriş ve çıkış ayarlarını belirler.
- **TIMx\_CCER (Capture/Compare Enable Register)**, Capture/compare kanallarını etkinleştirme veya devre dışı bırakma işlemlerini kontrol eder.
- **TIMx\_CNT (Counter Register)**, Timer'in ana sayıç değerini içerir. Bu register, zamanlayıcının sayma işlemini temsil eder.
- **TIMx\_PSC (Prescaler Register)**, Timer'in ön bölücü prescaler değerini içerir. Bu değer, timer'in sayma hızını kontrol eder.
- **TIMx\_ARR (Auto-Reload Register)**, Timer'in otomatik yeniden başlatma değerini içerir. Bu değer, sayacın bir döngü tamamlandığında otomatik olarak tekrar başlamasını sağlar.
- **TIMx\_CCR1 (Capture/Compare Register 1)**, Capture/compare modunda kullanılan karşılaşıştırma değerini içerir. Bu değer, belirli bir zaman noktasında veya karşılaşıştırma olayında kullanılır.
- **TIMx\_OR (Option Register)**, Timer'in özel seçeneklerini kontrol eder. Bu register, özel özelliklerin etkinleştirilmesi veya devre dışı bırakılması için kullanılır.

## Basic Timer

### TIM6, TIM7

- TIM6 ve TIM7 Basic Timer birimleri genel sayıç olarak kullanılabilecekleri gibi, spesifik olarak DAC biriminin tetikleyicisi olarak da kullanılabilirler.
- 16-bit genişliğinde auto-reload upcounter yani otomatik geri yüklenen artan sayaca sahiptir.
- 16-bit genişliğinde kontrol edilebilir prescaler değere sahiptir.
- DAC birimi için tetikleme çıkışlarına sahiptir.
- Interrupt ve DMA üretimi mevcuttur.
- Çalışma prensibi genel amaçlı timer'ların çalışma prensibi ile aynıdır.



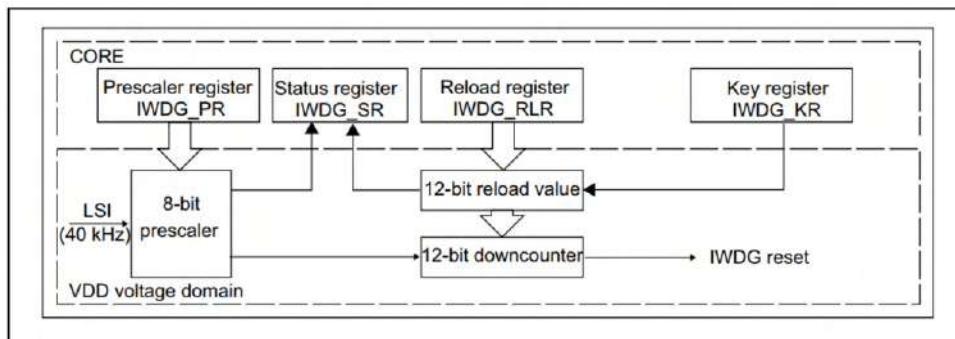


- **TIMx\_CR1 (Control Register 1)**, Timer'in genel kontrol ayarlarını içerir. Örneğin, Timer'in etkinleştirilmesi, zamanlama modu seçimi, otomatik yeniden başlatma etkinleştirme gibi ayarlar bu register üzerinden yapılmaktadır.
  - **TIMx\_CR2 (Control Register 2)**, dış tetikleyici konfigürasyonları gibi timer'in belirli özelliklerini ayarlamayı sağlar.
  - **TIMx\_DIER (DMA/Interrupt Enable Register)**, DMA ve interrupt izinlerini kontrol eder. Belirli olayların tetiklenmesi durumunda bir kesme talebi veya DMA transferi başlatma gibi işlevleri etkinleştirir veya devre dışı bırakır.
  - **TIMx\_SR (Status Register)**, bir taşıma durumu overflow olup olmadığını veya bir karşılaştırma olayının gerçekleşip gerçekleşmediğini belirtir.
  - **TIMx\_EGR (Event Generation Register)**, Olayların elle tetiklenmesini sağlar. Bu register üzerinden bir olayı event hemen tetikleyebilirsiniz.
  - **TIMx\_CNT (Counter Register)**, Timer'in ana sayaç değerini içerir. Bu register, zamanlayıcının sayma işlemini temsil eder.
  - **TIMx\_PSC (Prescaler Register)**, Timer'in prescaler değerini içerir. Bu değer, timer'in sayma hızını kontrol eder.
  - **TIMx\_ARR (Auto-Reload Register)**, Timer'in otomatik yeniden başlatma değerini içerir. Bu değer, sayacın bir döngü tamamlandığında otomatik olarak tekrar başlamasını sağlar.

## **Independent Watchdog (IWDG)**

- IWDG, işlemci saatinden bağımsız, kendine ait dahili RC osilatörden (LSI 32 KHz) beslenen bir watchdog timeridir.
  - Mikrodenetleyici içerisindeki amacı da bekçilik yapmaktadır. Mikrodenetleyici, harici sebeplerden veya kodlardaki bir hata sebebiyle kilitlenebilir. Mikrodenetleyici kilitlendiğinde, yürüttüğü işlemler durur. Bu tür durumlarda mikrodenetleyicinin tekrar başlatılması gereklidir. İşte watchdog timerlar burada devreye girerler. Watchdog timerlarda belirlenen bir süre sonunda sıfırlanırlar ve işlemciyi resetlerler.

Kullanıcı tarafından yapılmış bir kodun, mikrodenetleyicinin çalışmasını durdurması, programın çalışmaması, ya da başka bir nedenle devreye girdiğinde bu tür durumlarda mikrodenetleyicinin tekrar başlatılması gereklidir. İşte watchdog timerler burada devreye girerler. Watchdog timerlarda belirlenen bir süre sonunda sıfırlanırlar ve işlemciyi resetlerler.

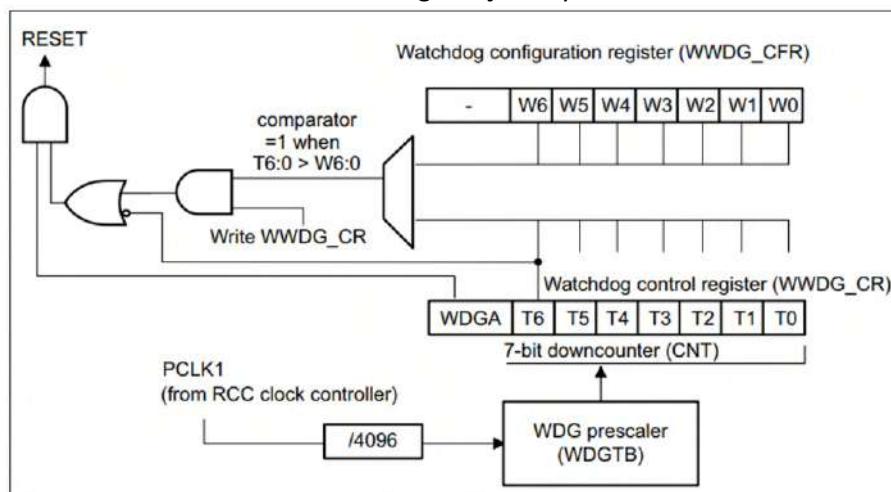


Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	IWDG_KR	Reserved												KEY[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x04	IWDG_PR	Reserved												PR[2:0]																			
	Reset 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			
0x08	IWDG_RLR	Reserved												RL[11:0]																			
	Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
0x0C	IWDG_SR	Reserved												RVU																			
	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			

- IWDG\_KR (Key Register)**, IWDG'yi kontrol etmek için kullanılan anahtar değerleri içerir. İlgili anahtar değerleri yazılarak IWDG'nin başlatılması, yeniden başlatılması veya durdurulması gibi işlemler gerçekleştirilebilir.
- IWDG\_PR (Prescaler Register)**, IWDG'nin zamanlayıcı değerini belirlemek için kullanılır. Zamanlayıcı değeri, bu ön bölücü ile çarparak IWDG'nin zamanlamasını elde eder.
- IWDG\_RLR (Reload Register)**, IWDG'nin zamanlayıcı değerini reload value içerir. IWDG'nin çalışması sırasında bu değer zaman içinde azalır, eğer bu değer sıfıra ulaşırsa, IWDG bir reset sinyali üretir.
- IWDG\_SR (Status Register)**, IWDG'nin durumunu gösteren bilgiler içerir. Örneğin, zaman aşımı durumu gibi bilgiler burada bulunabilir.

## Window Watchdog (WWDG)

- WWDG birimi belirli bir pencere içerisinde counter kaydedicisine tekrar değer yüklenебildiği için bu isimle anılmaktadır.
- Ayarlanabilir süre penceresine sahiptir.
- Anormal erken ve anormal geç uygulama davranışını algılayabilir.
- Önceden belirlenen duruma göre işlemciyi resetler.



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
0x00	WWDG_CR	Reserved												T[6:0]																			

0x00	<b>WWDG_CR</b>	Reserved		WDGA	T[6:0]							
					0	1	1	1	1	1	1	
0x04	<b>WWDG_CFR</b>	Reserved		EWI	WDGTB1	W[6:0]						
					WDGTB0	0	0	0	1	1	1	1
0x08	<b>WWDG_SR</b>	Reserved		EWIF								
					0							

- **WWDG\_CR (Control Register)**, WWDG'nin temel kontrol ayarlarını içerir. Özellikle, WWDG'nin etkinleştirilmesi, zamanlayıcı değeri (down-counter) ayarlanması ve bir reset talep biti bulunmaktadır.
- **WWDG\_CFR (Configuration Register)**, WWDG'nin daha fazla konfigürasyon ayarlarını içerir. Örneğin, window modunu etkinleştirme, zaman aşımı değeri ve window değeri gibi ayarları içerir.
- **WWDG\_SR (Status Register)**, WWDG'nin durumunu gösteren bilgiler içerir. Örneğin, zaman aşımı durumu ve window durumu gibi bilgiler burada bulunabilir.

# 07 PWM

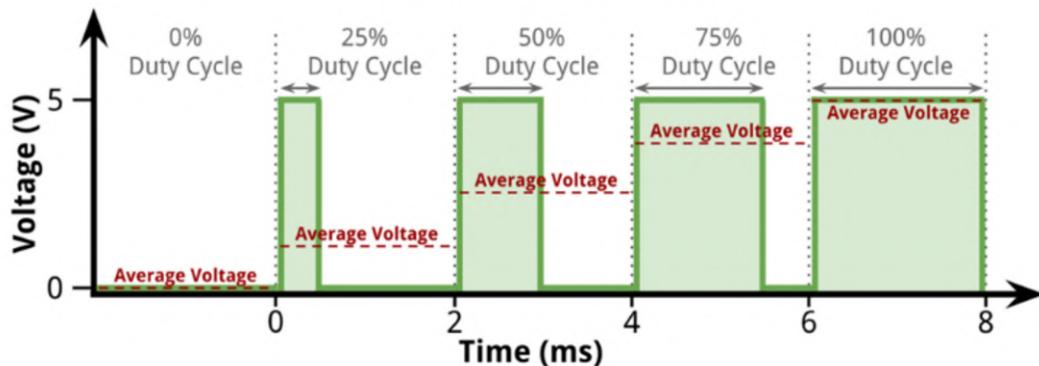
25 Haziran 2021 Cuma 23:48

## 07 PWM

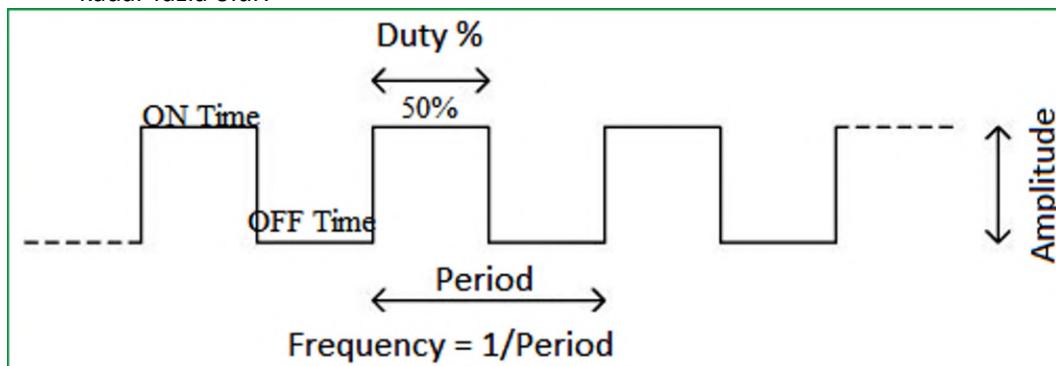
### Giriş

- <https://www.aydinlatma.org/pwm.html>, <https://berkannaydin.medium.com/pwm-nedir-5d20287970b5> linklerdeki makaleleri okuyabiliriz.
- PWM, Pulse Width Modulation (Darbe Genişlik Modülasyonu) bir kare dalga sinyalinin, yüksek seviyede kalma süresine müdahale ederek, bu sinyalin gerilimin ortalama değerinin değiştirilmesi olarak tanımlanabilir.,
- PWM endüstride iletişim, motor kontrol, ısıtma, aydınlatma gibi önemli bir çok alanda kullanılmaktadır.

### Pulse Width Modulation



- PWM, ışık kaynağını hızlı bir şekilde açık kapatarak parlaklığını ayarlamayı sağlayan bir modülasyon çeşididir.
- Anahtarlama işleminde açık kalma süresi ne kadar yüksek olursa yükle sağlanan güç yani ışık parlaklığı o kadar fazla olur.



- PWM teknlığında açık ve kapalı süresi görev döngüsü yani duty cycle ile tanımlanır. Ton açık süreyi, Toff kapalı süreyi temsil eder.
- Pulse Width, Ton süresi kadardır. Period, Ton ile Toff sürelerin toplamıdır.
- Duty Cycle aşağıdaki formül ile hesaplanır.

$$\text{Duty Cycle} = \frac{T_{on}}{T_{on} + T_{off}} * 100$$

- Giriş voltaj değeri ile Duty cycle değerini çarparak Ortalama Çıkış Gerilimini hesaplıyoruz.
- Frekans ise aşağıdaki formül ile hesaplanır. Frekans birimi Hz, Periyot birimi s'dir.

$$f = \frac{1}{T}$$

- PWM frekansını hesaplamak için, aşağıdaki formüllerden yararlanmamız lazım;  
Period = (Timer\_Tick\_Freq / PWM\_Freq) -1  
PWM\_Freq = Timer\_Tick\_Freq / (Period + 1)  
Timer\_Tick\_Freq = Timer\_CLK / (Prescaler + 1)
- Buradan şunu düşünmeliyiz. Timer frekansı kullanıcı tarafından belirlenir. Aynı zamanda PWM de istenilen frekansta çalışılacağı düşünülecek olursa, bizim belirleyeceğimiz iki değer var. Bunlardan biri prescaler, diğeri ise period. Aslında temel olarak PWM in istenilen frekansta çalışması için prescaler değeri küçük bir

değer seçilir ve period bu degere göre ayarlanır.

- **Mod 1**, Yukarı doğru sayarken CNT < CCRX (Capture Compare Register) dan düşükse kanal aktif, diğer durumda pasif olur. Aşağı doğru sayarken CNT > CCRX ise kanal pasif, değilse aktif olur.
- **Mod 2**, Yukarı doğru sayarken CNT < CCRX (Capture Compare Register) dan düşükse kanal pasif, diğer durumda aktif olur. Aşağı doğru sayarken CNT > CCRX is kanal aktif, değilse pasif olur.

# 08 UART

5 Mayıs 2021 Çarşamba 08:03

## 08 UART

### Giriş

- <https://cenntceylnn.medium.com/elektronik-haberleşme-protokollerinin-nedir-d11a6d3a5957> link üzerinden haberleşme protokollerini hakkında bilgi alabiliriz.
- <https://www.ercankoclar.com/2018/04/uart-iletisim-protokolu-ve-mikroc-kutuphanesi/>  
<https://arduinodestek.com/uart-haberlesme-nedir-ve-nasıl-gerçeklesir/>
- <https://youtu.be/uktFwZX2TTE>, <https://youtu.be/K0JuUAQYsaQ> ve <https://youtu.be/UCXVFJSrlbE> protokol hakkında videolardan bilgi edinebiliriz.
- <https://youtu.be/GRmYKJgAtQ4>, <https://youtu.be/NDwpWbXJ0sc>, <https://youtu.be/vrSzdoKv558>,  
[https://youtu.be/vzRuLn\\_Gzx8](https://youtu.be/vzRuLn_Gzx8), <https://youtu.be/ic8NUSytU-g>, <https://youtu.be/y7ZETFlohp0>,  
<https://youtu.be/1IGm99He7g4> linklerinden STM32 ile yapılmış örnek uygulamaları izleyebiliriz.
- UART (Universal Asynchronous Receiver Transmitter), 1 ve 0'lardan oluşan verileri iki dijital sistem arasında alıp verme işlemlerinde kullanılan bir iletişim protokolüdür.

### Avantajları ve Dezavantajları

Avantajlar;

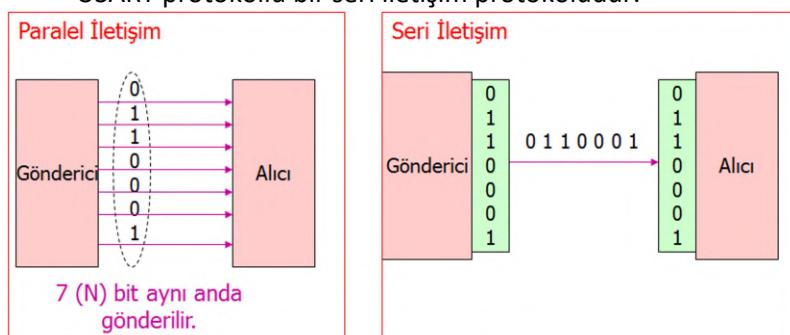
- Sadece iki kablo kullanır.
- Saat sinyali gereklidir.
- Hata denetimine izin vermek için bir eşlik biti vardır.
- Veri paketinin yapısı, her iki taraf da buna göre ayarlandığı sürece değiştirilebilir.
- İyi belgelenmiş ve yaygın olarak kullanılan yöntem.

Dezavantajlar;

- Veri çerçevesinin boyutu maksimum 9 bit ile sınırlıdır.
- Birden çok bağımlı veya birden çok ana sistemi desteklemez.
- Her UART'ın baud hızı, birbirinin %10'u dahilinde olmalıdır.

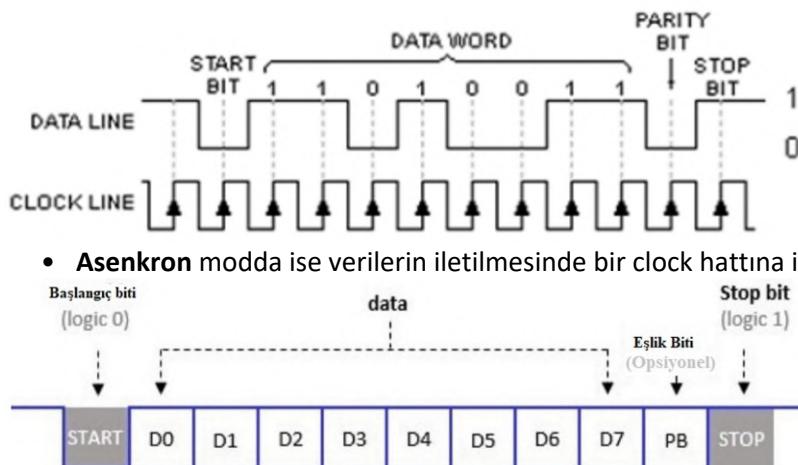
### İletişim Yöntemi

- Dijital sistemlerde iletişim paralel ve seri olmak üzere iki türlü yapılır.
- **Paralel** iletişimde bilgi vericiden alıcıya aynı anda birden fazla bit gidecek şekilde gönderilir. Böylece tek hamlede birden fazla veri karşı tarafa gönderildiği için iletişim hızı yüksektir. Fakat bu iletişim türünde kullanılan hat sayısı fazladır ve uzun mesafeler için uygun değildir.
- **Seri** iletişimde ise vericiden alıcıya gönderilecek bilgi, tek hat üzerinden sırayla gönderilir. Bu şekilde giden bilginin, tek hamlede tek biti gönderebileceği için iletişim hızı yavaşlatır. Fakat seri iletişimde hat sayısı azdır ve uzun mesafeli iletişim için daha uygundur.
- USART protokollü bir seri iletişim protokolüdür.

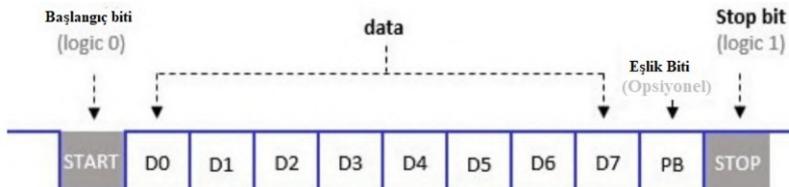


### İletişim Modu

- USART protokolünde veriler senkron veya asenkron olarak alınabilirler.
- **Senkron** veri alışverişinde bir data hattı ve bir clock hattı bulunmalıdır.  
Daha hattından gidecek veriler clock hattından gönderilen sinyalin her düşen veya yükselen kenarında alıcıya ilettilir.



- Asenkron modda ise verilerin iletilmesinde bir clock hattına ihtiyaç duyulmaz.

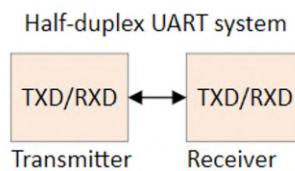
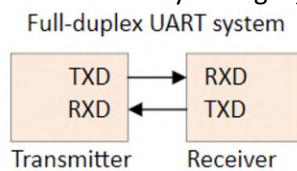
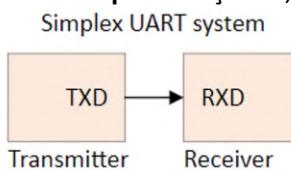


## Parametreler

- Verilerin gönderilmeye başlayacağı, alıcıya bir başlangıç için **Start** sinyali ile bildirir ve hemen arkasından veriler akmaya başlar daha sonrasında **Stop** biti ile sonlandırılır.
- Start biti "0" stop biti "1" verilerinden oluşmaktadır. Veri bitleri ise 7 veya 8 bit olabilir.
- Verilerin doğru olarak gönderilip gönderilmemiğini anlamak için kullanılan **Parity** biti bulunmaktadır. Parity bitinin gönderilmesi şart değildir.
- Parity biti genellikle üç farklı şekilde kullanılır. Even, Odd ve None olarak seçilebilir. Parite, Even olarak kullanıldığında, iletilen verinin toplamda çift sayıda yüksek seviyeye sahip olması gerekmektedir. Eğer iletilen verinin yüksek seviyeye sahip bit sayısı tek ise, parite biti 1 olacak şekilde ayarlanır ve toplamda çift sayıda yüksek seviye olmasını sağlar. Eğer iletilen verinin yüksek seviyeye sahip bit sayısı zaten çift ise, parite biti 0 olarak ayarlanır. Odd ise bunun tam tersidir. Parite bitinin kullanılmadığı durumlarda None seçeneğini kullanırız ve böylece iletilen verinin doğruluğu kontrol edilmez. Parite biti için boş bir bit alanı bırakılır ve sadece veri bitleri iletimi gerçekleştirilir.
- Parite biti, basit bir hata kontrol mékanizmasıdır ve veri bütünlüğünü sağlamak için kullanılır. Ancak, parite biti tek başına tüm hataları tespit etmek veya düzeltmek için yeterli değildir. Daha güvenli ve sağlam hata kontrol yöntemleri için farklı yöntemler ve algoritmalar kullanılabilir, örneğin CRC (Cycle Redundancy Check).
- Baudrate** saniyede gönderilen bit sayısıdır ve bps (bits per second - saniyede gönderilen bit sayısı) birimi ile ölçülür. Standart veri gönderme hızları 110, 150, 300, 600, 1200, 2400, 4800, 9600, 56000, 115200 gibi hızdadır. Baudrate hızı arttıkça veri iletim mesafesi azalır.
- Bir bitin iletimi için gereken süre, 1 / Baudrate olarak hesap edilir. Saniye cinsinden sonuç verir.
- Örneğin 9600 baud rate ile haberleşme yapıyorsak saniyede 9600 bit yani 1200 byte veri gönderebiliyoruzdur. Bir bitin iletimi için geçen süreyi 1/9600'den çıkan sonucu daha sonra us çevirmek için 1000000 ile çarparız. Sonuç olarak 104,16us bulunur. 115200 baudrate kullanımında 8,68us bulunur.
- İki cihaz arasında UART haberleşmesi yapılıyorsa, her iki cihazın da aynı baud oranı, veri bitleri, parite biti ve stop bitleri yapılmasına sahip olması gerekmektedir. Bu parametrelerin doğru bir şekilde yapılması, güvenilir ve hatasız veri iletimini sağlar.

## Çalışma Modları

- UART'ın üç çalışma modu vardır. Bunlar Full Duplex, Half duplex ve Simplex'dır.
- Full Duplex** iletişimde, veri gönderme ve veri alma işlemleri aynı anda ve bağımsız olarak gerçekleştirilir.
- Half Duplex** iletişimde, veri gönderme ve veri alma işlemleri aynı hattı paylaşan cihazlar arasında sırayla gerçekleştirilir.
- Simplex** iletişimde, veri iletimi sadece bir yönde gerçekleştirir.

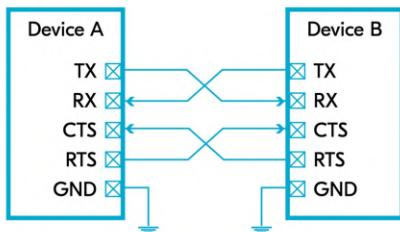


## Pin Yapısı

- TX (Transmit) ve RX (Receive), UART haberleşmesinde kullanılan veri gönderme ve veri alma hatlarını temsil eder.
- CTS (Clear To Send) ve RTS (Request To Send) ise UART haberleşmesinde kullanılan kontrol sinyalleridir. RTS ve CTS sinyalleri, veri akışını kontrol etmek için kullanılır ve genellikle aşırı yüklenmeyi önlemek veya veri

kaybını engellemek için kullanılır.

- **TX hattı**, veri gönderici tarafından kullanılan seri veri gönderme hattını temsil eder. Veri gönderici, veri paketini seri olarak TX hattına gönderir ve bu hattı kullanarak veriyi alıcıya iletir.
- **RX hattı**, veri alıcı tarafından kullanılan seri veri alma hattını temsil eder. Veri alıcı, veriyi seri olarak RX hattından alır ve bu hattı kullanarak veriyi işler.
- Veri gönderici tarafından veri göndermeye hazır olduğunu bildirmek için **RTS** sinyalini HIGH seviyeye çeker. Bu, veri alıcının veriyi almasını ve işlemeye başlamasını sağlar.
- Veri alıcı tarafından veri almayı ve işlemeyi kabul etmek için hazır olduğunu belirtmek için **CTS** sinyalini HIGH seviyede tutar. Bu, veri göndericinin veriyi göndermeye başlamasını sağlar.
- RX giriş pini ile TX çıkış pini ve CTS giriş pini ile RTS çıkış pini birbirlerine ters bağlanır.

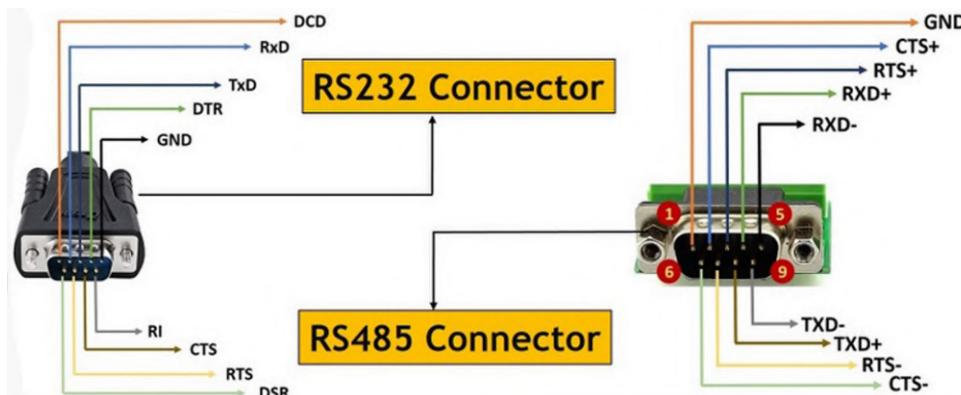


## Fiziksel Standartlar

- USART, seri iletişimde geniş bir kullanım alanına sahiptir ve çeşitli protokollerin uygulanmasına olanak tanır. Bu protokoller arasında RS232, RS422, RS485 fiziksel standartlar yer alır.
- <https://blog.direnc.net/rs232-ve-rs485-nedir-kullanim-alani-avantaj/>, <https://www.elektrikde.com/rs-232-rs-485-ve-rs-422-seri-iletisim/> ve <https://youtu.be/ChCRIU2kEEO> link üzerinden fiziksel standartlar hakkında bilgi edinebiliriz.
- **RS232**, tek bir veri iletim hattı üzerinden iletişim sağlar ve asenkron veri iletimine uygun bir protokol kullanır. Genellikle 15 volt ile -15 volt arasında bir gerilim seviyesi kullanır. Seri iletişim yaklaşık 15 metre kadardır.
- **RS-485 ve RS-422**, her ikisi de seri haberleşme standardı olan ve diferansiyel sinyal iletimini kullanan protokollerdir. RS485, RS422'nin bir üst kümesidir, bu nedenle tüm RS422 cihazları RS485 tarafından kontrol edilebilir.
- RS485, birden fazla cihaz arasında noktadan noktaya veya çok noktaya bağlantılar sağlayabilir. Düşük hızlardan (baud hızı) yüksek hızlara kadar geniş bir veri iletim hızı aralığına sahiptir. Genellikle 100 kbps ila 10 Mbps arasında değişen hızlarda iletişim sağlar. Uçtan uca maksimum mesafe 1200 metreye kadar olabilir.
- RS232 ile RS485 standartlarının özellikleri şu şekildedir:

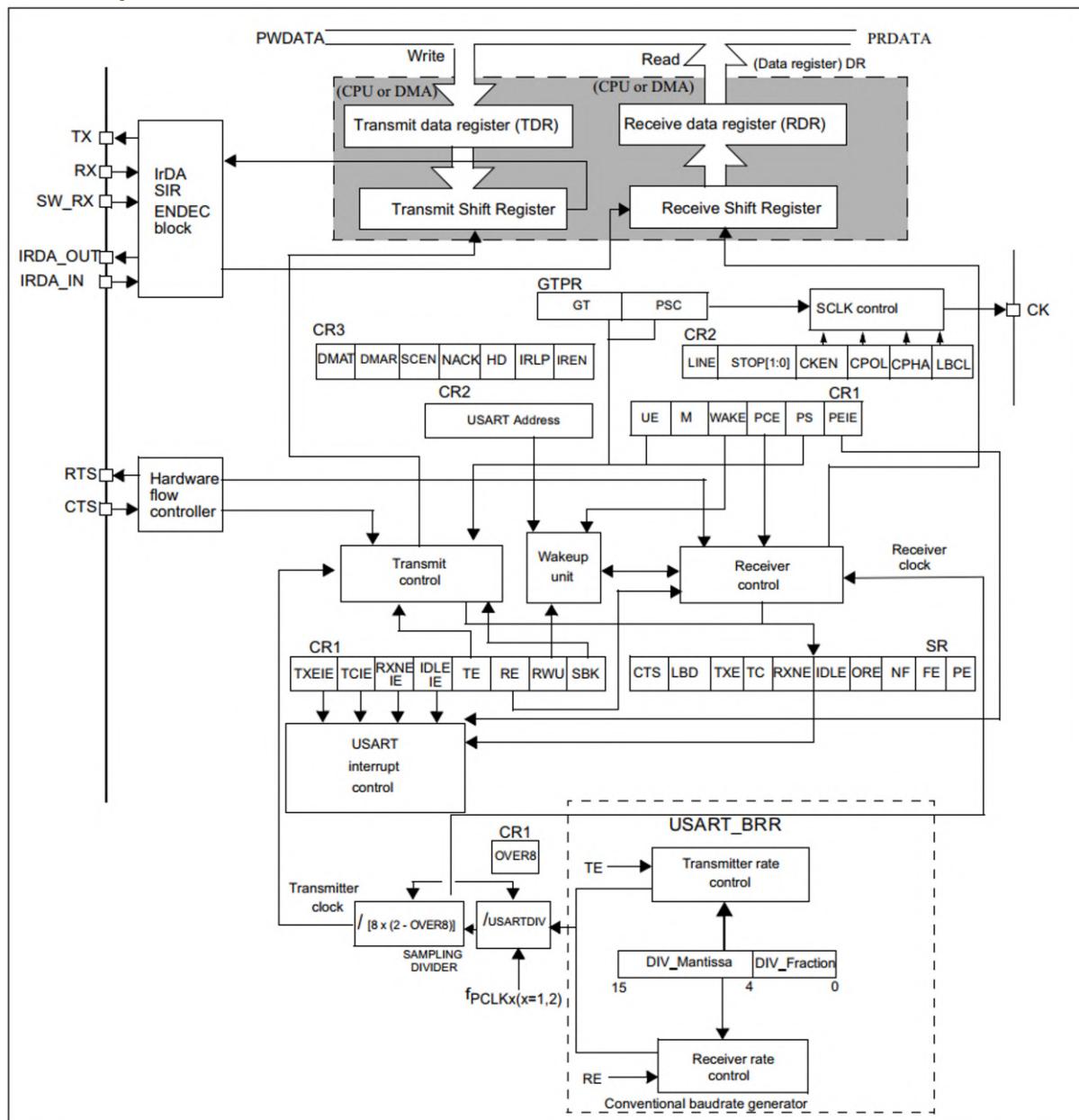
	RS232	RS485
Voltaj Sistemi	Gerilim seviyesine dayalı	Diferansiyel
Tek Hatta Toplam Sürücü ve Alıcı	1 Sürücü, 1 Alıcı	32 Sürücü, 32 Alıcı (Aynı anda bir Sürücü etkin)
Hat Yapılandırması	Noktadan Noktaya	Çok Aktarmalı
Maksimum Operasyonel Mesafe	15M/50FT	1200M / 3000FT
Maksimum Veri İletim Hızı	1 MBit/sn	10 MBit/sn
Maksimum Sürücü Çıkış Voltajı	±25V	-7V ila +12V
Alıcı Giriş Direnci	3 ila 7 kΩ	12 kΩ
Alıcı Giriş Voltaj Aralığı	±15V	-7V ila +12V
Alıcı Duyarlılığı	±3V	±200mV

- RS232 ve RS485 konnektörlerinin pin bağlantı bilgileri şu şekildedir:



- RS-232, eski ve yaygın olarak kullanılan bir seri haberleşme standartıdır. Ancak daha yüksek veri hızları, uzun mesafe iletimi ve çok noktalı haberleşme gerektiren uygulamalarda RS-485 gibi daha modern haberleşme standartları tercih edilmektedir.

## Birim Yapısı



## 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
0x00	USART_SR	Reserved															CTS	LBD TXE TC RXNE IDLE ORE NF FE PE															
	Reset value																0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
0x04	USART_DR	Reserved															DR[8:0]																
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x08	USART_BRR	Reserved										DIV_Mantissa[15:4]										DIV_Fraction [3:0]											
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x0C	USART_CR1	Reserved										OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	0	0	0	0	0	
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x10	USART_CR2	Reserved										LINEN	STOP [1:0]	CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBTL	Reserved	0	0	0	0	0	0	0	0	0	0		
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x14	USART_CR3	Reserved										ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEEN	NACK	HDSEL	IRLP	IREN	EIE	0	0	0	0	0	0	0	0	0	
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x18	USART_GTPR	Reserved										GT[7:0]								PSC[7:0]													
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

- **USART\_SR (Status Register)**, Bu kayıt, iletişim durumu hakkında bilgi sağlar. Örneğin, veri alımı veya iletimi tamamlandığında veya hata durumlarında bayraklar (flag) içerir.
- **USART\_DR (Data Register)**, iletişimde iletilen veya alınan veriyi tutar. Veriyi bu kayıta yazarak iletimi başlatabilir veya bu kayıttan okuyarak alınan veriyi elde edebilirsiniz.
- **USART\_BRR (Baud Rate Register)**, iletişim hızını ayarlamak için kullanılır.
- **USART\_CR1 (Control Register 1)** ve **USART\_CR2 (Control Register 2)**, UART modunu, iletim ve alım parametrelerini ve diğer iletişim ayarlarını kontrol eder.
- **USART\_CR3 (Control Register 3)**, üçüncü kontrol kaydıdır ve DMA ayarları gibi daha gelişmiş ayarları kontrol eder.
- **USART\_GTPR (Guard Time and Prescaler Register)**, zamanlama ayarları için kullanılır.

## Haberleşme Metotları

- UART üzerinden Polling, Interrupt ve DMA olmak üzere üç farklı haberleşme yapılabılır.
- <https://deepbluembedded.com/how-to-receive-uart-serial-data-with-stm32-dma-interrupt-polling/>, <https://controllerstech.com/uart-receive-in-stm32/> ve <https://controllerstech.com/uart-transmit-in-stm32/> link ile bu metotlar ile yapılan örnekleri inceleyebiliriz.
- **Polling** yani Yoklama yöntemi, mikrodenetleyici tarafından sürekli olarak UART veri alımını kontrol etmek için kullanılır. Mikrodenetleyici, UART veri alımını düzenli aralıklarla sorgular ve yeni veri varsa onu işler. Veri gelene kadar mikrodenetleyici diğer işlemleri yapamaz ve sürekli olarak UART'ı kontrol etmek zorunda kalır. Bu yöntem, basit uygulamalarda ve düşük hızlı veri iletiminde tercih edilebilir.
- Yalnızca UART kullanıyorsak ve başka bir şey kullanmıyorsak bu kullandığımız polling yöntemi kullanmak iyidir, aksi takdirde diğer tüm işlemler etkilenecektir.
- **Interrupt** yöntemi, UART'dan veri alındığında veya veri gönderildiğinde mikrodenetleyiciye kesme sinyali göndererek mikrodenetleyicinin normal işlemesini kesmesini sağlar. Bu yöntem, mikrodenetleyicinin sürekli olarak UART'ı sorgulamaktan kurtulmasını sağlar ve daha etkili bir şekilde diğer görevlerini gerçekleştirmesine olanak tanır. Interrupt yöntemi, yüksek hızlı veri iletimi veya zamanlama hassasiyeti gerektiren uygulamalarda tercih edilir.
- **DMA** yöntemi, veri transferini mikrodenetleyicinin müdahalesi olmadan doğrudan bellekten yapılmasını sağlar. DMA denetleyici, UART'dan gelen veya UART'a gönderilecek verileri doğrudan bellekten okur veya belleğe yazar. Bu yöntem, mikrodenetleyicinin UART veri transferiyle uğraşmadan diğer işlemleri gerçekleştirmesine olanak sağlar ve veri transferinde yüksek hızlı ve verimli bir çözümddür. DMA yöntemi, yüksek hızlı veri transferlerinde veya sürekli veri akışı gerektiren uygulamalarda kullanılır.
- Özellikle USART ile gönderilecek yüklü miktarda verimiz var ise, bu veriyi döngü içerisinde göndermek, işlemci zamanının önemli bir bölümünü harcayacaktır. Baud hızımız ne kadar az ise, gönderme hızımız o

kadar düşecek, dolayısıyla bekleme hızımız da o kadar artacaktır.

- Gönderme işleminin bitmesini beklemeden işimize devam edebilmek için ya DMA ya da EXTI kullanırız.
- Interrupt kullanımında ise CPU tarafından saniyede çok sayıda kesinti yapılması gerekecektir. Bu yüzden çok etkili yöntem değildir. Verileri doğrudan belleğe yönlendirmek için DMA biriminin kullanılması en etkili yöntemdir.

# 09 SPI

5 Mayıs 2021 Çarşamba 08:03

## 09 SPI

### Giriş

- <https://ozdenercin.com/2019/02/01/spi-seri-haberlesme-protokolu/>, <https://arduinodestek.com/spi-haberlesme-protokolu-nedir-ve-nasıl-gerçekleşir/> <https://devreyakan.com/spi-nedir/> linkinden ayrıntılı bilgilere ulaşabiliriz.
- SPI (Serial Peripheral Interface), mikrodenetleyiciler, sensörler, dijital IC'ler ve diğer entegre devreler arasında seri veri iletişimini için kullanılan bir seri senkron iletişim protokolüdür
- Özellik ve kullanım olarak I2C'ye benzer. I2C'de olduğu gibi bir adet Master cihaz bulunur. Bu cihaz hatta bağlı çevresel cihazları kontrol eder.
- Çevresel cihazlarla veya diğer mikrodenetleyicilerle veri transferi sağlayan yazılım/donanım tabanlı seri iletişim protokolüdür. Bu haberleşme şekli karşılıklı iki tarafın **clocklarının senkronize çalışmasıyla** data iletişimini sağlamaktadır.
- SPI'da veri transfer hızı I2C veri yolundan daha hızlıdır.
- SPI, genellikle düşük maliyetli, düşük güç tüketimi gerektiren uygulamalarda kullanılır. Özellikle mikrodenetleyiciler, sensörler, veri dönüştürücüler, hafıza kartları ve diğer entegre devreler arasında veri iletişimini için tercih edilir.
- SPI'nin hızlı ve basit bir iletişim protokolü olması, çeşitli uygulama alanlarında yaygın olarak kullanılmasını sağlar.

### Avantajları ve Dezavantajları

Avantajlar;

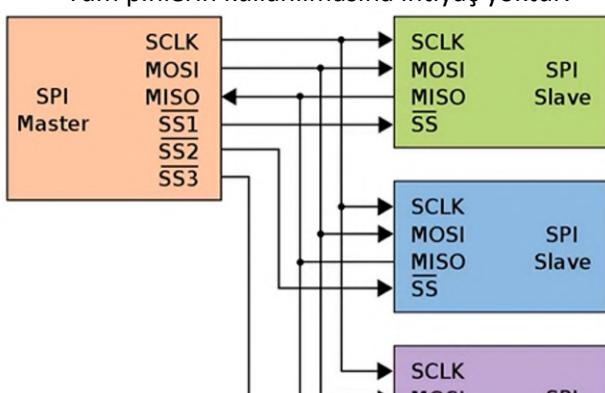
- Başlatma ve durdurma biti yok, böylece veriler kesintisiz olarak aktarılabilir.
- I2C gibi karmaşık bağımlı adresleme sistemi yok.
- I2C'den daha yüksek veri aktarım hızı (neredeyse iki kat daha hızlı).
- Ayrı MISO ve MOSI hatları, böylece veri aynı anda gönderilebilir ve alınabilir.

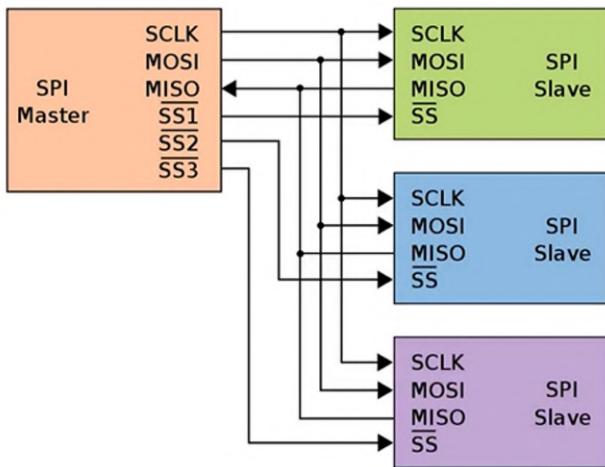
Dezavantajlar;

- Dört kablo kullanır (I2C ve UART'lar iki kablo kullanır).
- Verilerin başarıyla alındığına dair bir onay yok (I2C de vardır).
- UART'taki eşlik biti gibi hata denetimi biçimini yok.
- Yalnızca tek bir master'a izin verir.

### Bağlantılar

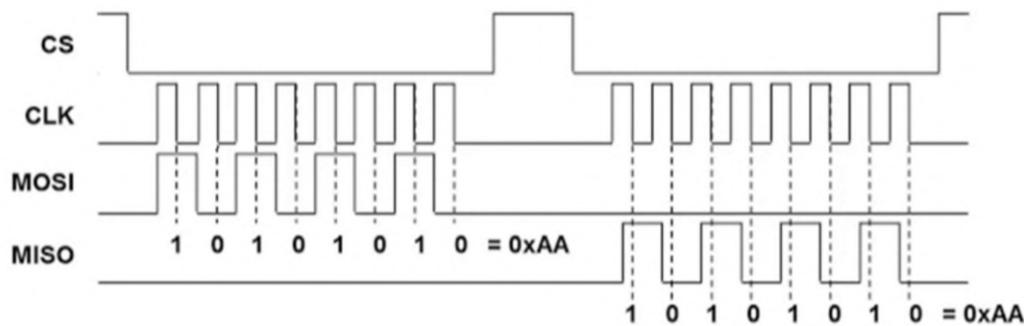
- SPI, genellikle dört telli bir bağlantıyla gerçekleştirilir:  
**MOSI** (Master Out Slave In), Master cihazdan (genellikle mikrodenetleyici) slave cihaza veri gönderir.  
**MISO** (Master In Slave Out), Slave cihazdan master cihaza veriyi gönderir.  
**SCLK** (Serial Clock): Saat sinyali, veri iletim hızını senkronize eder.  
Bu sinyal sadece master cihaz tarafından üretilir.  
**SS/CS** (Slave Select/Chip Select), iletişim kurulacak slave cihazı seçer.
- Slave cihaz donanımsal olarak seçildiği için I2C iletişimindeki gibi adres gönderilmez. Fakat birden fazla slave cihazın SPI veri yoluna bağlanması için birden fazla SS/CS pini kullanır.  
Tüm pinlerin kullanılmasına ihtiyaç yoktur.



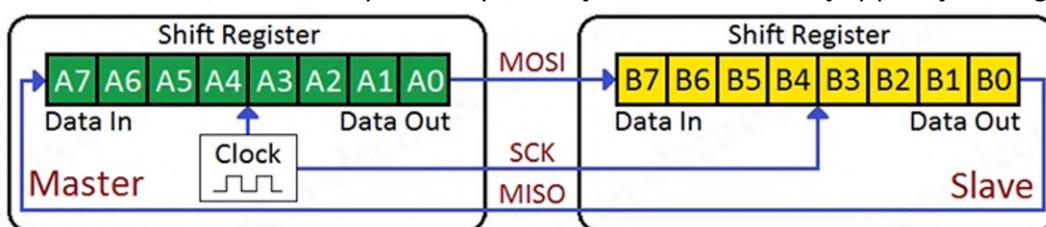


## Veri İletimi

- Master, saat sinyalini verir.
- Master, SS/CS pinini, slave etkinleştiren bir **LOW** voltaj durumuna geçirir.
- Master, verileri MOSI hattı boyunca her seferinde bir bit olarak slave'ye gönderir. Slave, bitleri alındıkça okur.
- Bir yanıt gerekiyorsa, bağımlı, MISO hattı boyunca her seferinde bir bit veriyi master'a döndürür. Master, bitleri alındıkça okur.



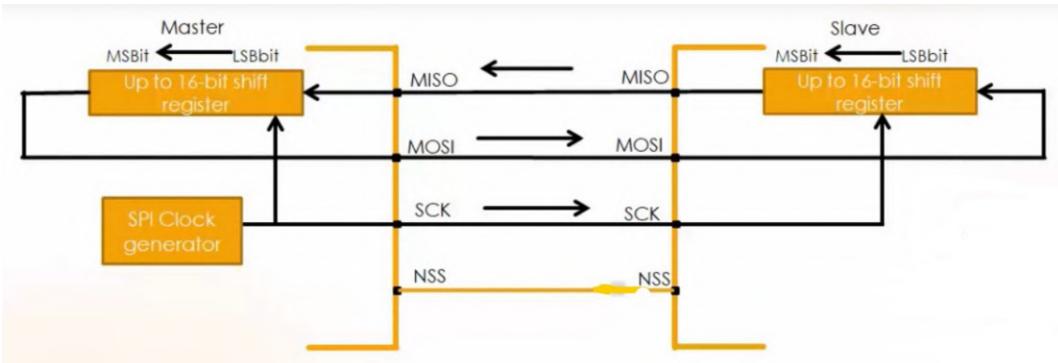
- SPI'da veri iletim sırası genellikle 8 bitlik veri paketleri halinde olur, ancak bu uzunluğu değiştirilebilir. Veri iletim sırası, verilerin en yüksek veya en düşük anlamlı bit ile başlayıp bitişebileceğinin şekilde yapılandırılabilir.



- Ana cihaz, iletişimi başlatır ve sonlandırır. Slave cihazlar ise ana cihazın taleplerine yanıt verir.

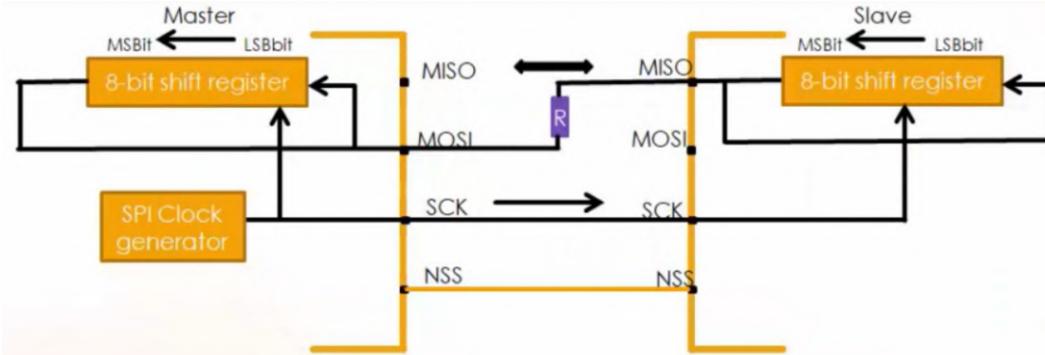
## Veri Yolu

- SPI haberleşmesi için üç farklı mod vardır. Bunlar Full Duplex, Half duplex ve Simplex'dır.
- İkişi çift yönlü iken diğeri tek yönlü haberleşmedir. Bu modlar hakkında detaylı bilgi almak için <https://fastbitlab.com/spi-bus-configuration-discussion-full-duplex-half-duplex-simplex/> linkteki yazıyı okuyabiliriz.
- Tek yönlü olan Simplex iletişiminde SS/CS pini kullanılmayabilir, ancak çift yönlü olan Full Duplex ve Half duplex iletişimde ve birden fazla slave cihazı varsa SS/CS pini kullanışlı olabilir.
- **Full Duplex**, hem veri gönderme hem de veri alma işlemlerinin **aynı anda** gerçekleştirir. Veri iletimi için iki ayrı iletişim hattı kullanılır. Her iki tarafta bağımsız olarak veri gönderebilir ve alabilir, bu nedenle iletişim hızı genellikle yüksektir.



- **Half Duplex**, sadece bir veri gönderme ya da bir veri alma işleminin aynı anda gerçekleştirildiği bir çalışma modudur.

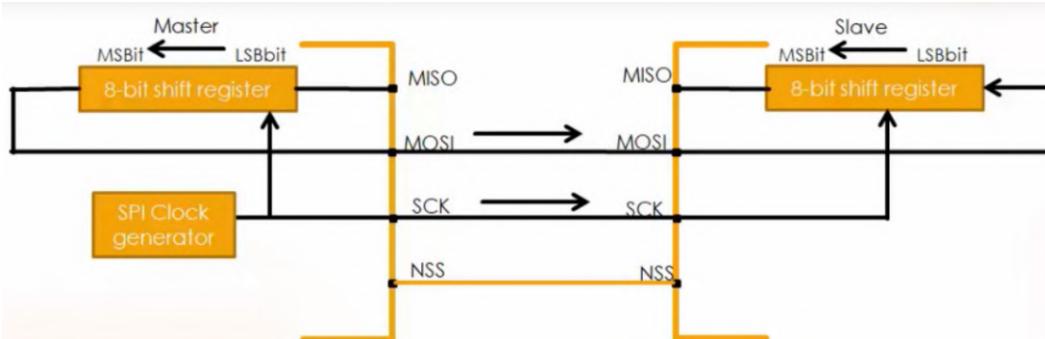
Bu modda, genellikle tek bir çift yönlü iletişim hattı kullanılır. Her iki taraf da aynı iletişim hattını kullanarak veri gönderebilir veya alabilir, ancak aynı anda her iki yönde veri iletimi yapılamaz.



- **Simplex**, sadece **bir yönde** veri iletimine izin veren bir çalışma modudur.

Genellikle tek bir iletişim hattı kullanılır ve veri gönderme veya veri alma işlemi yapılır. Her iki tarafta aynı anda veri gönderip alamaz.

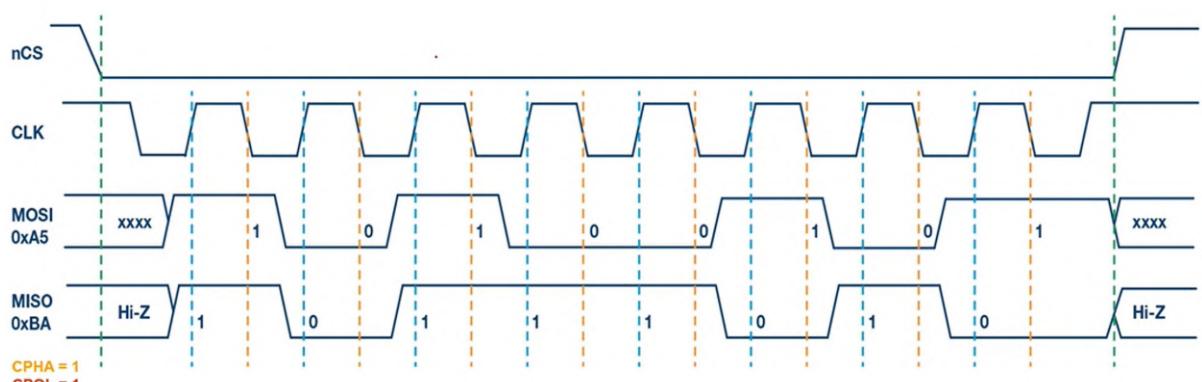
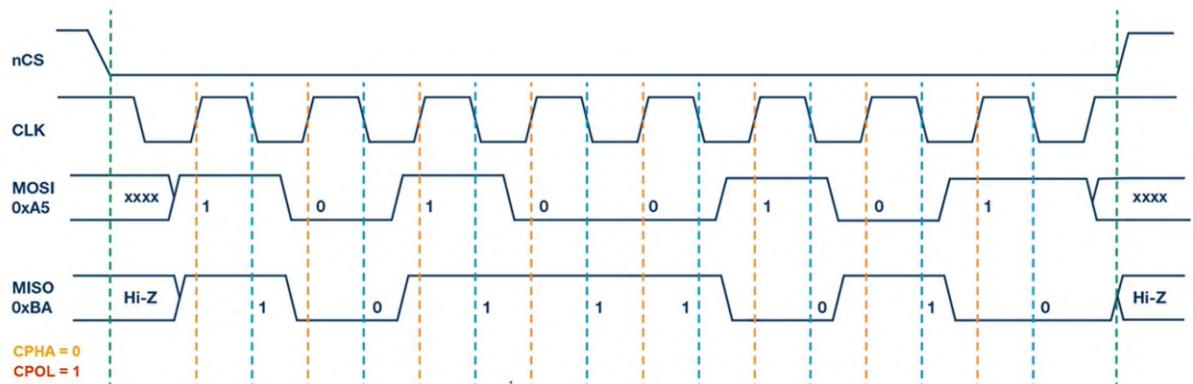
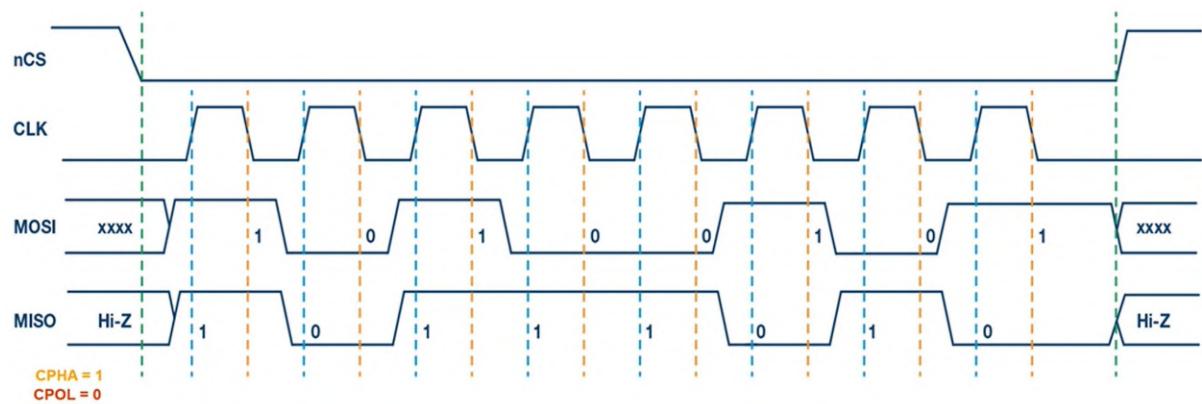
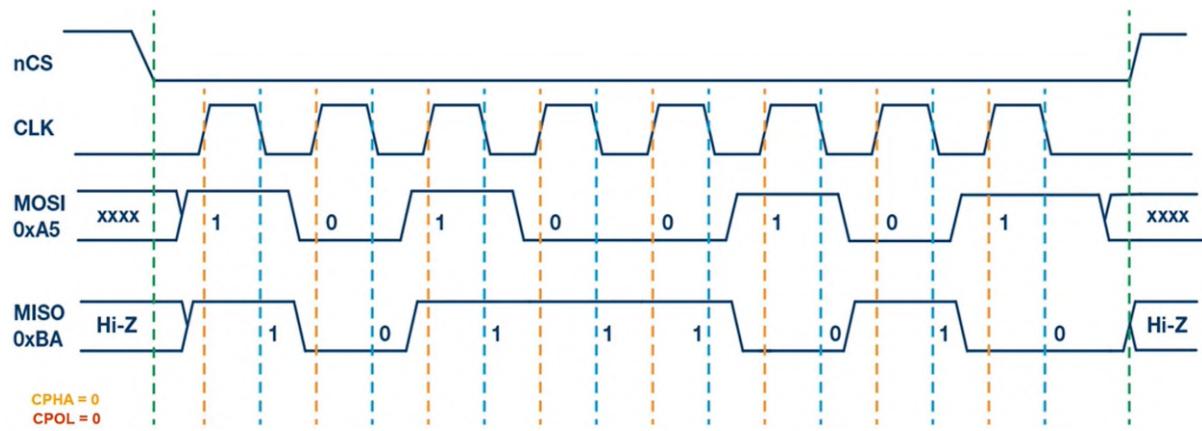
- Eğer master sadece veri gönderip slave sadece veri alacaksa, bu durumda MOSI hattı kullanılır. Diğer bir durumda, master sadece veri alıp slave sadece veri gönderecekse, bu durumda MISO hattı tercih edilir.
- Bu iletişimde SS/CS pini kullanılmayabilir. Çünkü Simplex modda genellikle tek yönlü iletişim olduğu için sadece master cihazın veri gönderme veya alım işlemlerini kontrol etmesi yeterlidir.



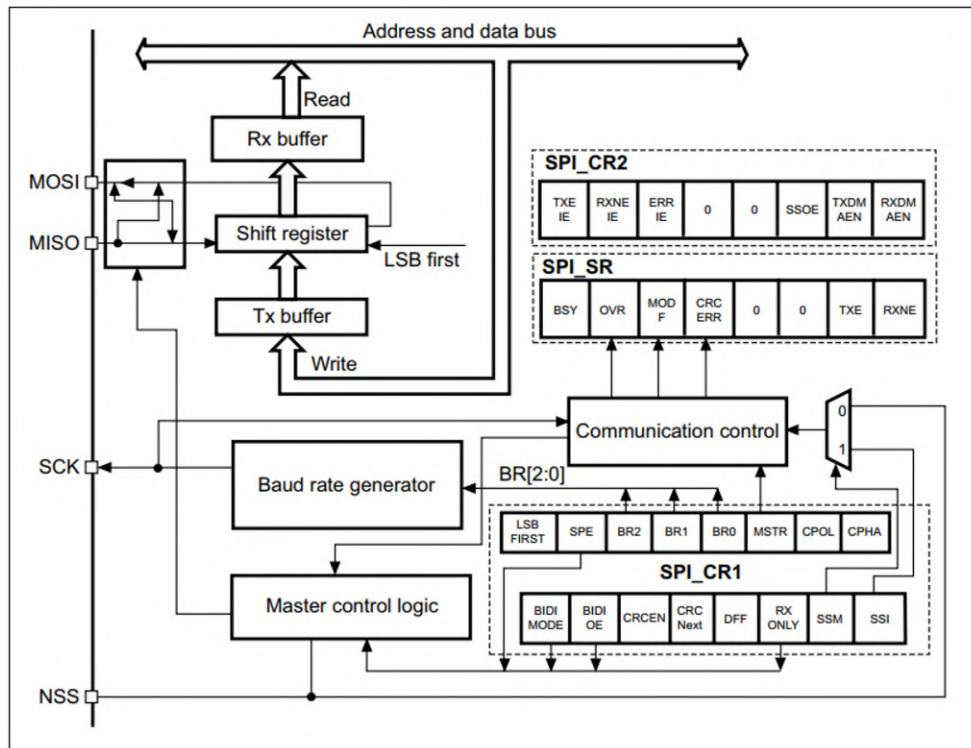
## Çalışma Modları

- SPI iletişiminde saat sinyalının nasıl üretileceğini belirler. Clock polarity (CPOL), saat sinyalının **yüksek** veya **düşük** seviyede başlayacağını belirtir, clock phase (CPHA) ise veri okuma/yazma işleminin saat sinyalinin hangi **kenarında** gerçekleşeceğini belirler.
- **SPI Modu 0 (CPOL=0, CPHA=0)**  
CPOL = 0: Saat sinyali, düşük seviyede başlar.  
CPHA = 0: Veri değişikliği saat sinyalının yükselen kenarında olur.
- **SPI Modu 1 (CPOL=0, CPHA=1)**  
CPOL = 0: Saat sinyali, düşük seviyede başlar.  
CPHA = 1: Veri değişikliği saat sinyalının düşen kenarında olur.
- **SPI Modu 2 (CPOL=1, CPHA=0)**  
CPOL = 1: Saat sinyali, yüksek seviyede başlar.  
CPHA = 0: Veri değişikliği saat sinyalının yükselen kenarında olur.
- **SPI Modu 3 (CPOL=1, CPHA=1)**  
CPOL = 1: Saat sinyali, yüksek seviyede başlar.

CPHA = 1: Veri değişikliği saat sinyalinin düşen kenarında olur.



## Birim Yapısı



## 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
0x00	SPI_CR1	Reserved												BIDIMODE	0	0	0	0	0	0	0	0	0	0	0	0	LSBFIRST	7	3	2	1	0	
	Reset value													BIDI OE	0	0	0	0	0	0	0	0	0	0	0	0	SPE	6	0	0	0	0	
0x04	SPI_CR2	Reserved												CRCEN	0	0	0	0	0	0	0	0	0	0	0	0	ERRIE	0	0	0	0	0	
	Reset value													CRCNEXT	0	0	0	0	0	0	0	0	0	0	0	0	FRF	0	0	0	0	0	
0x08	SPI_SR	Reserved												FRE	0	0	0	0	0	0	0	0	0	0	0	0	SSI	8	4	2	1	0	
	Reset value													TXEIF	0	0	0	0	0	0	0	0	0	0	0	0	LSBFIRST	7	3	2	1	0	
0x0C	SPI_DR	Reserved												TXNEIF	0	0	0	0	0	0	0	0	0	0	0	0	RXNEIE	0	0	0	0	0	
	Reset value													ERRIE	0	0	0	0	0	0	0	0	0	0	0	0	FRF	0	0	0	0	0	
0x10	SPI_CRCPR	Reserved												CRCPOLY[15:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value													UDR	0	0	0	0	0	0	0	0	0	0	0	0	CHSIDE	0	0	0	0	0	
0x14	SPI_RXCRCR	Reserved												RxCrc[15:0]	0	0	0	0	0	0	0	0	0	0	0	0	TxCrc[15:0]	0	0	0	0	0	
	Reset value													TxCrc[15:0]	0	0	0	0	0	0	0	0	0	0	0	0	RxCrc[15:0]	0	0	0	0	0	
0x18	SPI_TXCRCR	Reserved												TxCrc[15:0]	0	0	0	0	0	0	0	0	0	0	0	0	TxDmaen	0	0	0	0	0	
	Reset value													TxDmaen	0	0	0	0	0	0	0	0	0	0	0	0	Rxdmaen	0	0	0	0	0	

- **SPI\_CR1 (Control Register 1)** ve **SPI\_CR2 (Control Register 2)**, iletişim modunu (Master veya Slave) ve saat hızını, Data frame format, Half duplex iletişim ve diğer özelliklerini yapılandırmak için kullanılır.
- **SPI\_SR (Status Register)**, SPI haberleşme durumunu izlemek için kullanılır. İletimin tamamlanıp tamamlanmadığı, veri alımı durumu gibi bilgileri içerir.
- **SPI\_DR (Data Register)**, Veri gönderip almak için kullanılır. Gönderilen veya alınan veriyi bu kayıt aracılığıyla işleyebilirsiniz.
- **SPI\_CRCPR (CRC Polynomial Register)** ve **SPI\_RXCRCR/SPI\_TXCRCR (CRC Receive/Transmit Register)**, SPI verilerinin döngüsel hata denetimi (CRC) için kullanılır.

## Haberleşme Metotları

- SPI üzerinden Polling, Interrupt ve DMA olmak üzere üç farklı haberleşme yapılabilir.
- <https://deepbluembedded.com/stm32-spi-tutorial/> linkinden konu hakkındaki bilgileri inceleyebiliriz.

# 10 I2C

5 Mayıs 2021 Çarşamba 08:03

## 10 I2C

### Giriş

- <https://www.ercankoclar.com/2018/01/i2c-iletisim-protokolu-ve-mikroc-kutuphanesi/> ve <https://ozdenercin.com/2019/01/25/i2c-seri-haberlesme-protokolu/> linklerinden ayrıntılı bilgilere ulaşabiliriz.
- [https://www.ti.com/lit/an/slva704/slva704.pdf?ts=1702118996423&ref\\_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253DUnderstanding%2Bthe%2BI%2B2C%2BBus%2526nr%253D1136](https://www.ti.com/lit/an/slva704/slva704.pdf?ts=1702118996423&ref_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253DUnderstanding%2Bthe%2BI%2B2C%2BBus%2526nr%253D1136) linkten Texas Instruments'in I2C protokü hakkında yazdığı makaleyi okuyabiliriz.
- I2C protokolünün geliştirilme amacı, düşük hızlı çevre birimlerinin ana kartları, cep telefonları, gömülü sistemler gibi elektronik cihazlara daha az kablo ihtiyacı ile bağlanabilmesini sağlamaktadır.

### Avantajları ve Dezavantajları

Avantajlar;

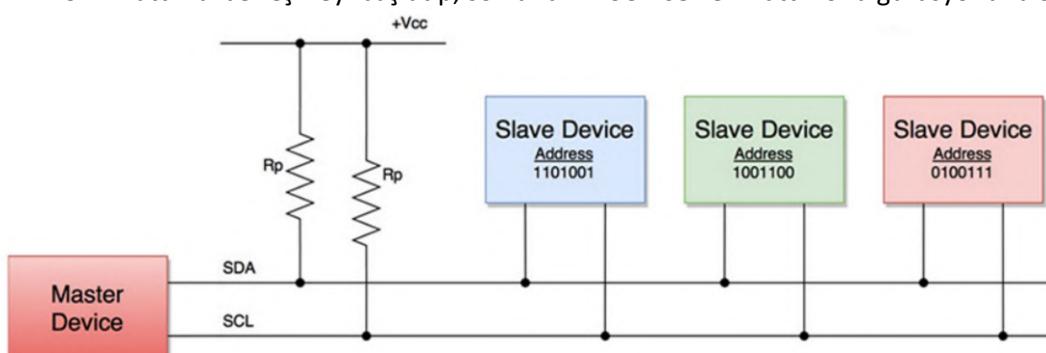
- Sadece iki telli bir yapıya sahiptir (SCL ve SDA), bu nedenle **az pin** kullanımını ile birçok cihazın bağlanmasıını sağlar.
- Aynı hat üzerinden **çift yönlü iletişim** sağlar. Hem master hem de slave cihazlar veri gönderebilir ve alabilir.
- Bir dizi cihazın (EEPROM, sensörler, ekranlar vb.) bağlanması destekler, bu da çok **çeşitli uygulamalara** olanak tanır.
- Master ve slave cihazlar arasındaki bağlantıları daha **esnek** hale getirir. Çoklu master bağlantılarına izin verir.
- **Open-drain çıkışı**, güç tüketimini azaltır ve daha güvenilir bir iletişim sağlar.
- **Yüksek veri iletim hızlarına** izin verecek şekilde tasarlanmıştır.

Dezavantajlar;

- I2C'nin **uzun hat mesafelerinde** performansı düşük olabilir. Bu durum, iletişim hızını düşürmek veya ek güçlendirme önlemleri almak gerektirebilir.
- Birden çok masterin bulunduğu sistemlerde **çatallanma** sorunları ortaya çıkabilir. Bu durum, çakışmaları önlemek için dikkatlice senkronize edilmiş bir sistem gerektirir.
- Başlangıçta **karmaşık** olabilir ve doğru yapılandırma ve senkronizasyon gerektirebilir.
- Önceden belirlenmiş bir adres yapısı kullanır, bu nedenle **güvenlik** açısından zayıf olabilir.
- Uzun hat mesafelerinde veya yüksek hızlarda iletişimde, **elektromanyetik girişim** sorunları ortaya çıkabilir.

### Bağlantılar

- I2C iletişiminde sadece iki hat vardır. Bunlar SDA (Serial Data Line) ve SCL ( Serial Clock Line) hatlarıdır.
- Genellikle +5V ve +3.3V voltajlarda çalışmakla beraber, I2C protokolü daha pratik voltaj seviyelerine de izin vermektedir.
- SDA hattı harberleşmeyi başlatıp, sonlandırır. SCL ise veri hattı konfigurasyonunu sağlar.

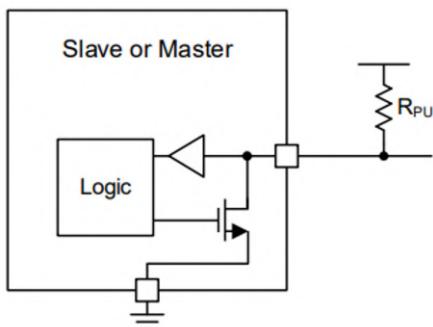


### Çift Yönlü Haberleşme

- I2C, aynı hat üzerinde open-drain/open-collector ile bir giriş buffer kullanır, bu da tek bir veri hattının çift yönlü veri akışı için kullanılmasına olanak tanır.
- Bu hatlar ayrıca **pull-up** direncine ihtiyaç duyarlar.
- Yalnızca bir cihaz veri yolu hattını toprağa çekebilir veya veri yolu hattını serbest bırakır ve böylece pull-up

direncinin voltajı yükseltmesine izin verir.

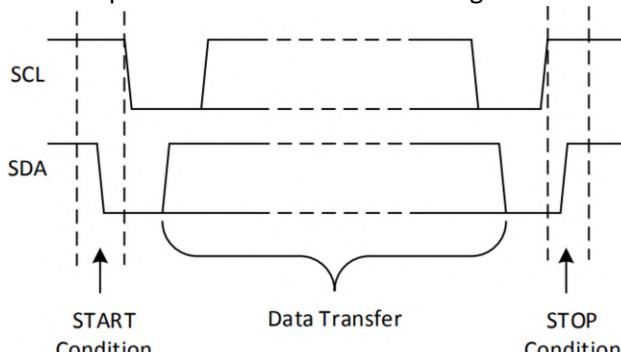
- Hattı LOW seviyeye çekmek için FET transistör **tetiklenir** böylece hat toprak ile kısa devre olur.
- Hattı HIGH seviyeye çekmek için FET transistör **kapatılır** böylece hat pull-up direnci vasıtıyla voltaj seviyesine çekilir.



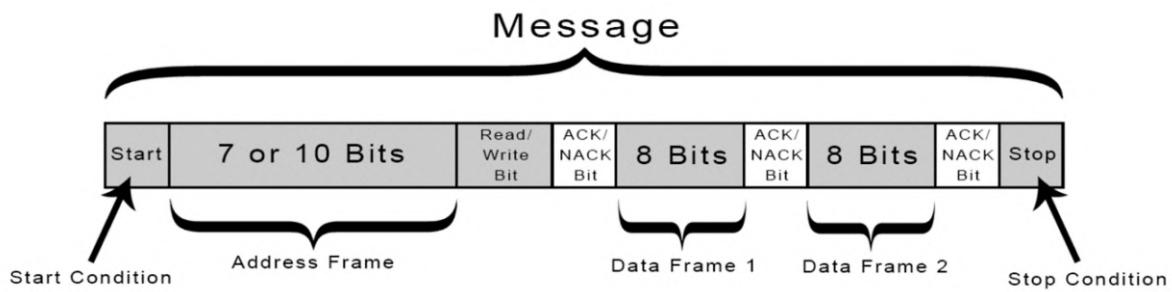
## Veri İletimi

- I2C hattı SDA hattının lojik high seviyesinden lojik low seviyeye düşmesi ile başlar. Aynı şekilde lojik low seviyesinden lojik high seviyeye çıkması ile sonlanır. SDA hattının haberleşmeyi başlatılabilmesi için SCL hattı da high olmalıdır.
- SCL hattı lojik high seviyesinde iken SDA hattı high seviyesine çekilirse haberleşme sonlanır.
- I2C veri gönderiminde start biti "0" stop biti "1" verilerinden oluşmaktadır.
- I2C veriyolu multimaster bir yapıdadır. Bu sayede iletişim hattında birden fazla cihaz olabilir. Master cihazlarda bir saat sinyali ve data gönderildiği anda diğer cihazların tamamı slave moduna geçerler.
- Multimaster I2C haberleşmesinde Repeated Start komutu vardır ve sıkılıkla kullanılır. I2C haberleşmesinde 2 adet cihaz olduğunu varsayıyalım.

Birinci master cihaz start komutu gönderdi ve start komutundan sonra gerekli adres bilgilerini gönderdi. Tüm bu işlemler sürecinde I2C hattı birinci master cihaz tarafından kullanıldığından dolayı I2C hattı idle durumda olmayacağıdır. Birinci cihaz stop durumu göndermeden önce haberleşmede bir değişiklik yapmak isterse Repeated Start komutunu gönderir ve böylece 1.Master cihazın slave cihaz ile I2C haberleşmesi kopmamış olur. Multimaster olmayan durumlarda Repeated Start komutunu kullanmaya gerek yoktur. Repeated Start komutu ard arda gelen start stop komutlarından oluşur.

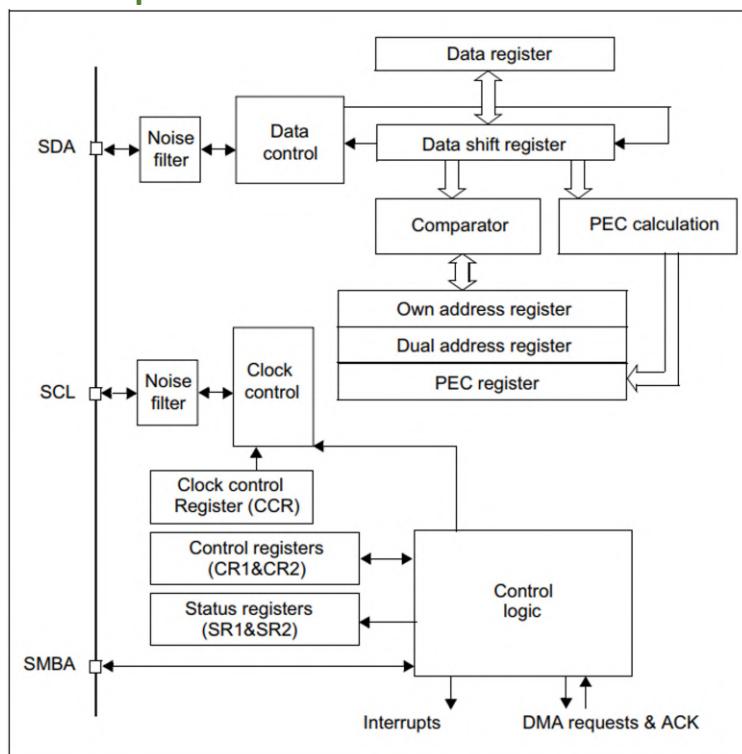


- İlk olarak SDA ve SCL hatları HIGH konumdadırlar. Daha sonra SDA hattı master tarafından **LOW** seviyeye çekilerek **iletişimin başlayacağı**, slave cihazlara bildirilir.
- Bu bildirimi alan slave cihazlar, adres bilgisini beklemeye başlarlar.
- Adres bilgisi** slave cihazların yapısına göre 7 bit, 10 bit veya 16 bit olabilirler.
- Master cihaz hangi slave cihaz ile haberleşmek istiyorsa onun adres bilgisini gönderdikten sonra, **okuma** mı yoksa **yazma** mı yapacağını belirtir. Adres hangi slave cihazın ise o cihaz master ile iletişim kurmaya başlar.
- Adres kendisine ait olan slave cihaz, master cihaza verinin gönderildiği veya verinin alındığını doğrulamak için bir **ACK** (Acknowledge) kabul biti gönderir.
- Veri transferi işlemi gerçekleşir. Bu transfer iki yönlü de olabilir. (Slave'den Master'a veya Master'dan Slave'e)



- I2C haberleşmesinde 1 master cihaz ve birden fazla slave cihaz olduğunu varsayılmı. Master cihaz herhangi bir slave cihaza erişmek için start komutundan sonra ilgili slave cihazın adresini gönderir. Aynı hatta bağlı olan slave cihazların tamamı bu mesajları alır ancak sadece bu mesaja sahip olan slave cihaz Ack mesajını göndererek iletişim kurulduğu master cihaza bildirir ve Ack mesajını alan master cihaz adres bilgisinden hemen sonra veri göndermeye başlar.

## Birim Yapısı



## Register

- **I2C\_CR1 (Control Register 1)**, Ana kontrol registeridir. I2C Peripherals'ı etkinleştirir veya devre dışı bırakır. Gönderme tamamlandığında kesme (interrupt) etkinleştirir. Acknowledge kontrol biti ile Yazılım sıfırlama biti bulunmaktadır.
  - **I2C\_CR2 (Control Register 2)**, ikinci kontrol registeridir. Saat frekansını belirler.
  - **I2C\_OAR1 (Own Address Register 1)**, I2C'nin kendi adresini ayarlamak için kullanılır.
  - **I2C\_DR (Data Register)**, veri göndermek veya almak için kullanılır.
  - **I2C\_SR1 ve I2C\_SR2 (Status Register 1 ve 2)**, I2C'nin durumu hakkında bilgi sağlar. Birçok farklı durumu içerir, örneğin, START biti durumu, adres gönderme durumu, veri alım durumu vb.
  - **I2C\_CCR (Clock Control Register)**, I2C saat frekansını kontrol eder.
  - **I2C\_TRISE (Rise Time Register)**, yükselme süresini ayarlamak için kullanılır.
  - **I2C\_FLTR (Filter Register)**, I2C hatlarında gürültüyü azaltmaya yönelik bir filtreleme mekanizması sağlar.

## **Haberleşme Metotları**

- SPI üzerinden Polling, Interrupt ve DMA olmak üzere üç farklı haberleşme yapılabılır.
  - <https://deepbluembedded.com/stm32-i2c-tutorial-hal-examples-slave-dma/> linkinden konu hakkındaki bilgileri inceleyebiliriz.

# Harici Led Yakma

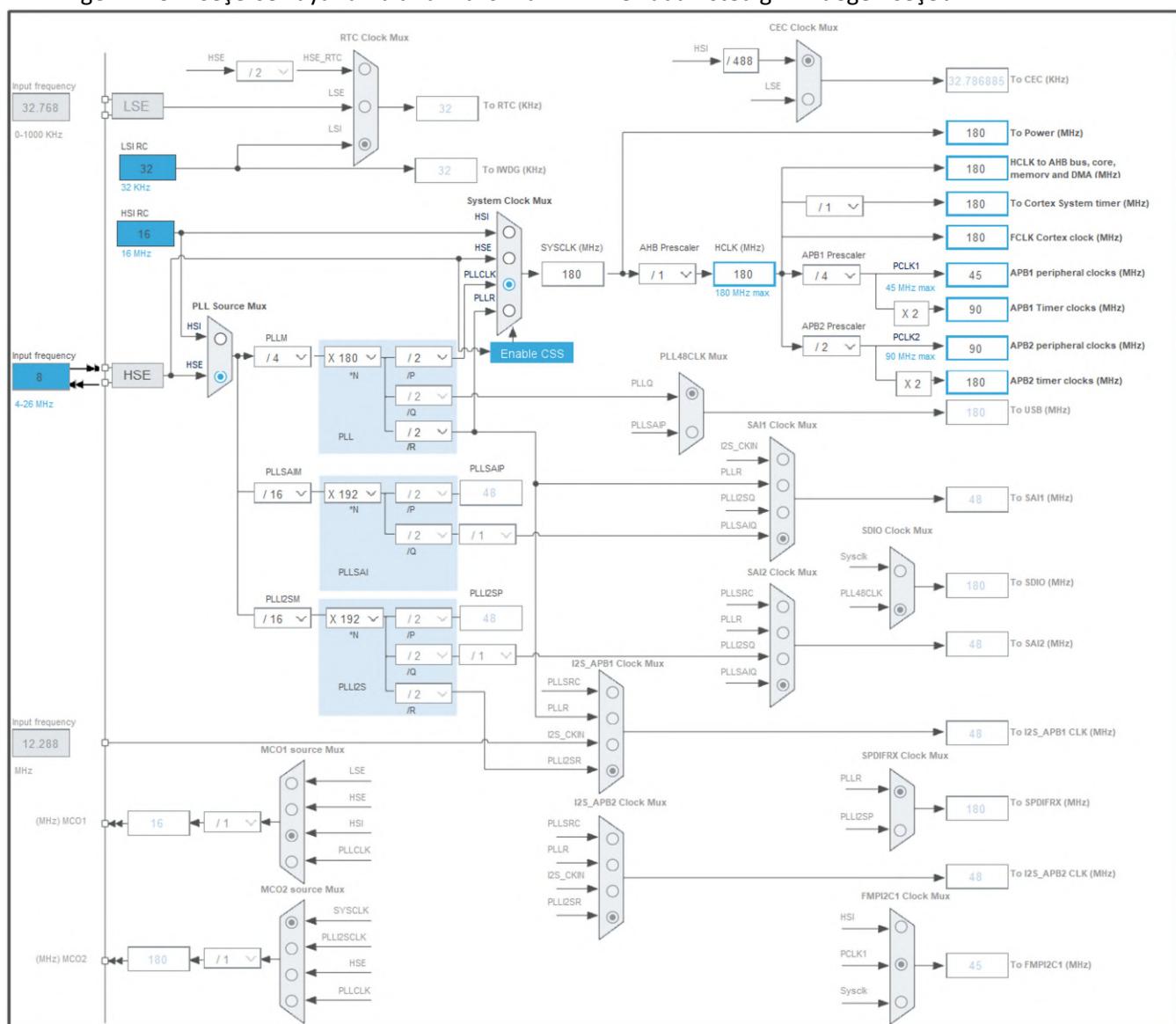
25 Aralık 2021 Cumartesi 00:52

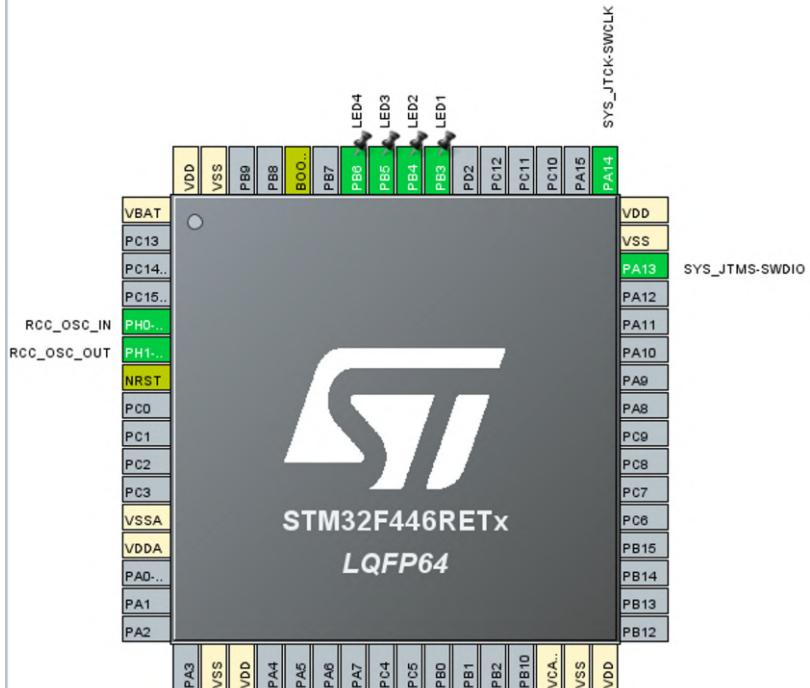
## Harici Led Yakma

### ➤ HAL

#### Konfigürasyon Kısmı

- Bir saniye içerisinde ne kadar talimat çalıştırıysa frekansı o kadardır.
- Sistem saatini ayarlamak için öncelikle RCC kısmından HSE'den Crysal/Ceramic Resonator seçilir. Bu seçim işlemi ile beraber Clock Configuration da kutucuk bölme açılır. LSE'nin HSE ile aralarındaki farkı düşük hızlı olması ve düşük güç tüketimi için kullanılır.
- System Clock Mux kısmında 3 seçenekimiz var. Bunlar HSI, HSE ve PLLCLK'tur. HSI dahili iken HSE harici kaynaktır.
- HSI seçersek kutucuktaki değer olur, HSE seçersek giriş frekansı belli MHz aralıktadır. Biz burada kart üzerinde 8MHz olduğundan bu şekilde yapıp kullanıyoruz.
- Eğer PLLCLK seçersek ayarlamalarla maksimum MHz'e kadar istediğimiz değeri seçebiliriz.





- System Core kısmından SYS tıklıyoruz ve Debug'da Serial Wire diyoruz.

Pin Name	Signal on Pin	GPIO output ...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA13	SYS_JTMS-SWDIO	n/a	n/a	n/a	n/a		<input type="checkbox"/>
PA14	SYS_JTCK-SWCLK	n/a	n/a	n/a	n/a		<input type="checkbox"/>

- RCC tıklıyoruz HSE'de Crystal/Ceramic Resonator işaretliyoruz.

Pin Name	Signal on Pin	GPIO output ...	GPIO mode	GPIO Pull-u...	Maximum ou...	User Label	Modified
PH0-OSC_IN	RCC_OSC_IN	n/a	n/a	n/a	n/a		<input type="checkbox"/>
PH1-OSC_OUT	RCC_OSC_OUT	n/a	n/a	n/a	n/a		<input type="checkbox"/>

- B portundaki 3., 4., 5. ve 6.pinlere led bağladık.

Pin Name	Signal	GPIO o...	GPIO mode	GPIO Pull-up/Pull-do...	Maximum output...	User Label	Modified
PB3	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED1	<input checked="" type="checkbox"/>
PB4	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED2	<input checked="" type="checkbox"/>
PB5	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED3	<input checked="" type="checkbox"/>
PB6	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED4	<input checked="" type="checkbox"/>

## Kod Kısmı

- CubeMX'de ayarlamaları yaptıktan sonra main.c kısmında yaptığımız kısımları kendisi kod içerisinde oluşturmuştur.
  - RCC ve GPIO için iki ayrı fonksiyon oluşturmuştur.
  - Oluşturulan iki fonksiyon kod içerisinde yazmadan önce private function prototypes kısmında belirtilir.

```
48 /* Private function prototypes */
```

```
49 void SystemClock_Config(void);
```

```
50 static void MX_GPIO_Init(void);
```

- Daha sonra bu iki fonksiyon kod içerisinde dahil edilir.

```

64@ int main(void)
65 {
66     /* USER CODE BEGIN 1 */
67
68     /* USER CODE END 1 */
69
70     /* MCU Configuration-----*/
71
72     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
73     HAL_Init();
74
75     /* USER CODE BEGIN Init */
76
77     /* USER CODE END Init */
78
79     /* Configure the system clock */
80     SystemClock_Config();
81
82     /* USER CODE BEGIN SysInit */
83
84     /* USER CODE END SysInit */
85
86     /* Initialize all configured peripherals */
87     MX_GPIO_Init();
88     /* USER CODE BEGIN 2 */
89
90     /* USER CODE END 2 */
91
92     /* Infinite loop */
93     /* USER CODE BEGIN WHILE */
94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         /* USER CODE BEGIN 3 */
99     }
100    /* USER CODE END 3 */
101 }

```

- hal\_gpio.c kısmından WritePin ve Toggle Pin fonksiyonlarına ulaşabiliriz.

```

410@void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
411 {
412     /* Check the parameters */
413     assert_param(IS_GPIO_PIN(GPIO_Pin));
414     assert_param(IS_GPIO_PIN_ACTION(PinState));
415
416     if(PinState != GPIO_PIN_RESET)
417     {
418         GPIOx->BSRR = GPIO_Pin;
419     }
420     else
421     {
422         GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;
423     }
424 }

```

```

433@void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
434 {
435     uint32_t odr;
436
437     /* Check the parameters */
438     assert_param(IS_GPIO_PIN(GPIO_Pin));
439
440     /* get current Output Data Register value */
441     odr = GPIOx->ODR;
442
443     /* Set selected pins that were at low level, and reset ones that were high */
444     GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
445 }

```

- Döngümüze aşağıdaki kodu yazarız.

```

94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         /* USER CODE BEGIN 3 */
99         HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin, GPIO_PIN_SET);
100        HAL_Delay(2000);
101        HAL_GPIO_TogglePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
102        HAL_Delay(500);

```

```

94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         /* USER CODE BEGIN 3 */
99         HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin, GPIO_PIN_SET);
100        HAL_Delay(2000);
101        HAL_GPIO_TogglePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
102        HAL_Delay(500);
103    }

```

### Programlama Kısmı

- STM32CubeIDE üzerinden hex dosyası oluşturmak için <https://www.youtube.com/watch?v=7FpsxwEXfLc> link üzerinden bilgi alabiliriz.
- Proje dosyasına sağ tıklayıp Properties tıklarız. Burada C/C++ Build sekmesinde Setting tıklarız ve burada Build Steps sekmesine geliriz. Burada Post-build steps kısmında Command kutucوغuna "arm-none-eabi-objcopy -O ihex \${ProjName}.elf \${ProjName}.hex" yazarız. Bu kod ile .elf dosyasını .hex dosyasına çeviririz.
- Hex dosyasını yüklemek için STM32 ST-LINK Utility ya da STM32Cube Programmer programlarını kullanabiliriz. Eğer STM32Cube Programmer programını kullanıyorsak kodu yükledikten sonra işlemci üzerindeki reset tuşuna basarak yüklediğimiz kod çalıştırırız.
- Eğer STM32Cube Programmer programını kullanıyorsak kodu yükledikten sonra işlemci üzerindeki reset tuşuna basarak yüklediğimiz kod çalıştırırız.

### Git Kısmı

- STM32CubeIDE üzerinden oluşturduğumuz projeyi git ile kullanmak için [https://www.youtube.com/watch?v=\\_96FSH7uIOE](https://www.youtube.com/watch?v=_96FSH7uIOE) linkteki videoya ve <https://shadyelectronics.com/how-to-use-github-with-stm32cubeide/> linkteki makaleye göz atabiliriz.

## ➤ REGİSTER

### Konfigürasyon Kısmı

- main.c dosyasındaki yorum satırları silip sadece hale getiriyoruz.

```

1 #include "stm32f4xx.h"
2
3 int main(void)
4 {
5     while (1)
6     {
7
8     }
9 }

```

- system\_stmf4xx.c dosyasındaki 182.satırda SystemCoreClock kısmına Ctrl ile sağ tıklıyoruz ve bizi system\_stmf4xx.h dosyasındaki 59.satırı götürüyor. Bu satırı kopyalayıp main.c dosyasına yapıştırıyoruz.

```
182     uint32_t SystemCoreClock = 168000000;
```

```
59 extern uint32_t SystemCoreClock;           /*!< System Clock Frequency (Core Clock) */
```

```

1 #include "stm32f4xx.h"
2
3 extern uint32_t SystemCoreClock;
4
5 uint32_t systemClock;
6
7 int main(void)
8 {
9     systemClock=SystemCoreClock;
10
11     while (1)
12     {
13
14     }
15 }
```

Expression	Type	Value
(*)= systemClock	uint32_t	168000000

```

1 #include "stm32f4xx.h"
2
3 extern uint32_t SystemCoreClock;
4
5 uint32_t systemClock;
6
7 int main(void)
8 {
9     systemClock=SystemCoreClock;           //168 000 000
10
11    RCC_DeInit();                      //HSI ON PLL OFF
12
13    SystemCoreClockUpdate();
14    systemClock=SystemCoreClock;         //16 000 000
15
16    while (1)
17    {
18
19    }
20 }

```

Expression	Type	Value
(x)= systemClock	uint32_t	16000000\

- STM32F407VG mikrodenetleyicinin Reference Manuals'de RCC kısmına bakıyoruz. Mikrodenetleyici 32 bittir. Bu sebeple kaydediciler olan registerler 32 bittir.
- Bit değerlerinde yazan r ile rw'nin anlamı r'nin read yani okunabilir, w'nin write yani yazılabılır anlamı vardır. r olan yerlere müdahale edemiyoruz ama rw yazan bitlere müdahale edebiliyoruz.

## RCC clock control register (RCC\_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved			PLL12S RDY	PLL12S ON	PLLRD Y	PLLON	Reserved			CSS ON	HSE BYP	HSE RDY	HSE ON			
			r	rw	r	rw				rw	rw	r	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HSICAL[7:0]								HSITRIM[4:0]						Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	r	rw		

- Adress offset bizim ilk olarak kaydedilen yeri gösterir.  
Reset value ise girilen değer sonrası tüm bitler sıfırlanır.
- Burada resetleme işlemi yapıyoruz.

RCC->CR &= 0x00000083;

- "&=" anlam kaydet anlamındadır.
- Harici osilatör kullanacağımdan HSEON olan 16.biti 1 yapmam gerekiyor.

Bit 16 **HSEON**: HSE clock enable

Set and cleared by software.

Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.

0: HSE oscillator OFF

1: HSE oscillator ON

- "|= anlamı ise öncekine 1 ekle ve eşitle anlamındadır. Bunu kullanırken eklemeye yani ötelemeye yapıyoruz.
- 16.biti 1 yapmak için ötelemeye kullanırken 1 << 16 yazıyoruz yani 0.bitte 1 yazıp 16 bit öteleiyor.

RCC->CR |= 1 << 16;

Bunu denetlemek için 17.biti kullanıyoruz. Yani 16.bit 1 olup olmadığını HSE osilatör 6 clock cycles yaptıktan sonra 17.bit olan HSERDY biti 1 oluyor.

Bit 17 **HSERDY**: HSE clock ready flag

### Bit 17 HSERDY: HSE clock ready flag

Set by hardware to indicate that the HSE oscillator is stable. After the HSEON bit is cleared, HSERDY goes low after 6 HSE oscillator clock cycles.

- 0: HSE oscillator not ready
- 1: HSE oscillator ready

- While döngüsü 1 olduğu sürece çalışır. HSERDY biti 1 olana dek 0 olacağından ve döngünün çalışabilmesi için başına "!" işaretini koyarak tersliyoruz.
- 17.bit 1 olduğunda döngüden çıkışacaktır.
- & biti lojik kapılarda olduğu gibi 0&0 ile 0&1 olduğunda çıkışında 0 verir. 1&1 olduğunda çıkışında 1 verir.
- Buradaki  $1 \ll 17$  işlemini yapabilmesi için 6 clock cycles zaman geçmesi gerekiyor. Öteleme işlem yaptığını anlamak için RCC->CR 1 olduğunda 17.bit 1 olmuş oluyor. Böylece 1&1'den 1 oluyor ve !1'den 0 olarak döngüden çıkışıyor.

```
while(!(RCC->CR & (1 << 17)));
```

- HSI kapatmak için HSION bitini 0 yapmam gerekiyor.

### Bit 0 HSION: Internal high-speed clock enable

Set and cleared by software.

Set by hardware to force the HSI oscillator ON when leaving the Stop or Standby mode or in case of a failure of the HSE oscillator used directly or indirectly as the system clock. This bit cannot be cleared if the HSI is used directly or indirectly as the system clock.

- 0: HSI oscillator OFF
- 1: HSI oscillator ON

- $1 \ll 0$  ile 0.bit 1 yapılır ardından ~ işaretini ile 0 yapılır.

```
RCC->CR &= ~(1 << 0)
```

- Şu an mikrodenetleyici 8 000 000Hz'de çalışıyor. PLL ile 168 000 000Hz'e çıkaracağız.

### Bit 19 CSSON: Clock security system enable

Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if an oscillator failure is detected.

0: Clock security system OFF (Clock detector OFF)

1: Clock security system ON (Clock detector ON if HSE oscillator is stable, OFF if not)

- 19.bit 1 yapılır.

```
RCC->CR |= 1 << 19;
```

PLLON 24.bit 1 yapılmadan önce PLL'nin konfigürasyonlarını yapmamız gerekiyor.

## RCC PLL configuration register (RCC\_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

Access: no wait state, word, half-word and byte access.

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLL_N / PLL_M)$
- $f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLL_P$
- $f_{(USB\ OTG\ FS,\ SDIO,\ RNG\ clock\ output)} = f_{(VCO\ clock)} / PLL_Q$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserved	PLLSRC	Reserved			PLL_P1	PLL_P0		
			rw	rw	rw	rw		rw				rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	PLLN								PLLM5	PLLM4	PLLM3	PLLM2	PLLM1	PLLM0	
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

- PLL için M, N ve P değerlerine göre sistem saatı 168MHz yapıyoruz. Bunun için M değerine 4, N değerine 168 ve P değerine 2 verdigimizde bu işlemi sağlamış oluyoruz.
- M için PLLM kısmında ilk 6 bitte yazacağımız 000100 değerini 0., 1., 3., 4. ve 5.bite 0 yapıyorken 2.biti 1 yapıyoruz.

2.biti 1 yaparken ekleme yaparak yani "|=" işaretini kullanırken diğerlerine sadece 0 bite kaydetmek için

"&=" işaretini kullanıyoruz.

Bits 5:0 **PLLM**: Division factor for the main PLL (PLL) and audio PLL (PLLI2S) input clock

Set and cleared by software to divide the PLL and PLLI2S input clock before the VCO.

These bits can be written only when the PLL and PLLI2S are disabled.

**Caution:** The software has to set these bits correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

VCO input frequency = PLL input clock frequency / PLLM with  $2 \leq PLLM \leq 63$

000000: PLLM = 0, wrong configuration

000001: PLLM = 1, wrong configuration

000010: PLLM = 2

000011: PLLM = 3

000100: PLLM = 4

...

111110: PLLM = 62

111111: PLLM = 63

```
RCC->PLLCFGR &= ~(1 << 0);      //PLLM0 0
RCC->PLLCFGR &= ~(1 << 1);      //PLLM1 0
RCC->PLLCFGR |= (1 << 2);       //PLLM2 1
RCC->PLLCFGR &= ~(1 << 3);      //PLLM3 0
RCC->PLLCFGR &= ~(1 << 4);      //PLLM4 0
RCC->PLLCFGR &= ~(1 << 5);      //PLLM5 0
```

- N için PLLN kısmını kullanıyoruz.

Bits 14:6 **PLLN**: Main PLL (PLL) multiplication factor for VCO

Set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when PLL is disabled. Only half-word and word accesses are allowed to write these bits.

**Caution:** The software has to set these bits correctly to ensure that the VCO output frequency is between 100 and 432 MHz.

VCO output frequency = VCO input frequency  $\times$  PLLN with  $50 \leq PLLN \leq 432$

00000000: PLLN = 0, wrong configuration

00000001: PLLN = 1, wrong configuration

...

000110010: PLLN = 50

...

001100011: PLLN = 99

001100100: PLLN = 100

...

110110000: PLLN = 432

110110001: PLLN = 433, wrong configuration

...

111111111: PLLN = 511, wrong configuration

*Note: Multiplication factors ranging from 50 and 99 are possible for VCO input frequency higher than 1 MHz. However care must be taken that the minimum VCO output frequency respects the value specified above.*

HEX	A8
DEC	168
OCT	250
BIN	1010 1000

- 168 decimal sayısının binary karşılığı 1010 1000'dir. Biz burada 9 bit olduğundan 010101000 yazacağız.  
Biz bununla uğraşmak yerine 6 bit öteleyip 168 sayısını yazacağız.

```
RCC->PLLCFGR |= (168 << 6);      //PLLN 168
```

Aynı işlemi M için de yapabiliriz.

```
RCC->PLLCFGR |= (4 << 0);      //PLLM 4
```

P için PLLP kısmını kullanıyoruz. 2 değeri için iki biti 0 yap diyor.

Bits 17:16 **PLLP**: Main PLL (PLL) division factor for main system clock

Set and cleared by software to control the frequency of the general PLL output clock. These bits can be written only if PLL is disabled.

**Caution:** The software has to set these bits correctly not to exceed 168 MHz on this domain.

PLL output clock frequency = VCO frequency / PLLP with PLLP = 2, 4, 6, or 8

00: PLLP = 2  
01: PLLP = 4  
10: PLLP = 6  
11: PLLP = 8

```
RCC->PLLCFGR &= ~(1 << 16);           //PLLP1 0
RCC->PLLCFGR &= ~(1 << 17);           //PLLP2 0
```

- PLL için başlangıçta tüm bitleri 0 yapıyoruz.
- Burada 8 tane sıfır var. Buradaki her biri 32 bit'de 4 bite karşılık geliyor.

```
RCC->CFGGR = 0x00000000;
```

- Tüm bitleri başlangıçta 0 yaptığımız için PLL'de P için yaptığımız 0'lama işlemine gerek kalmadı.

```
7 void RCC_Config(void)
8 {
9     //RCC->CR &= 0x00000083;           //RESET
10
11    RCC->CR &= ~(1 << 0);          //HSION
12    RCC->CR |= 1 << 16;             //HSEON
13    while(!(RCC->CR & (1 << 17))); //HSERDY
14    RCC->CR |= 1 << 19;             //CSSON
15    RCC->CFGGR = 0x00000000;
16    //RCC->PLLCFGR &= ~(1 << 0);    //PLLM0 0
17    //RCC->PLLCFGR &= ~(1 << 1);    //PLLM1 0
18    //RCC->PLLCFGR |= (1 << 2);      //PLLM2 1
19    //RCC->PLLCFGR &= ~(1 << 3);    //PLLM3 0
20    //RCC->PLLCFGR &= ~(1 << 4);    //PLLM4 0
21    //RCC->PLLCFGR &= ~(1 << 5);    //PLLM5 0
22    RCC->PLLCFGR |= (4 << 0);       //PLLM 4
23    RCC->PLLCFGR |= (168 << 6);     //PLLN 168
24    //RCC->PLLCFGR &= ~(1 << 16);   //PLLP1 0
25    //RCC->PLLCFGR &= ~(1 << 17);   //PLLP2 0
26 }
```

Registers		
Register	Address	Value
> DMA1		
v RCC		
> CR	0x40023800	0xb7083
v PLLCFGR	0x40023804	0xa04
1010 PLLQ3	[27:1]	0x0
1010 PLLQ2	[26:1]	0x0
1010 PLLQ1	[25:1]	0x0
1010 PLLQ0	[24:1]	0x0
1010 PLLSRC	[22:1]	0x0
1010 PLLP1	[17:1]	0x0
1010 PLLP0	[16:1]	0x0
1010 PLLN8	[14:1]	0x0
1010 PLLN7	[13:1]	0x1
1010 PLLN6	[12:1]	0x0
1010 PLLN5	[11:1]	0x1
1010 PLLN4	[10:1]	0x0
1010 PLLN3	[9:1]	0x1
1010 PLLN2	[8:1]	0x0
1010 PLLN1	[7:1]	0x0
1010 PLLN0	[6:1]	0x0
1010 PLLM5	[5:1]	0x0
1010 PLLM4	[4:1]	0x0
1010 PLLM3	[3:1]	0x0
1010 PLLM2	[2:1]	0x1
1010 PLLM1	[1:1]	0x0
1010 PLLM0	[0:1]	0x0

- PLLSRC ile PLL sürücüsü seçilir. Bu bit için HSI kullanırsak 0, HSE kullanırsak 1 yapılır. Biz HSE kullandığımızdan bu biti 1 yapıyoruz.

#### Bit 22 **PLLCSR**: Main PLL(PLL) and audio PLL (PLLI2S) entry clock source

Set and cleared by software to select PLL and PLLI2S clock source. This bit can be written only when PLL and PLLI2S are disabled.

0: HSI clock selected as PLL and PLLI2S clock entry

1: HSE oscillator clock selected as PLL and PLLI2S clock entry

```
RCC->CR |= (1 << 22);
```

#### Bit 24 **PLLON**: Main PLL (PLL) enable

Set and cleared by software to enable PLL.

Cleared by hardware when entering Stop or Standby mode. This bit cannot be reset if PLL clock is used as the system clock.

0: PLL OFF

1: PLL ON

```
RCC->CR |= 1 << 24;
```

#### Bit 25 **PLLRDY**: Main PLL (PLL) clock ready flag

Set by hardware to indicate that PLL is locked.

0: PLL unlocked

1: PLL locked

```
while(!(RCC->CR & (1 << 25)));
```

## RCC clock configuration register (RCC\_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]				
rw		rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Reserved	HPRE[3:0]				SWS1	SWS0	SW1	SW0	
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	r	r	rw	rw	

- İlk önce 0.bit 0 yapıldı daha sonra 1.bit 1 yapılarak sistemin saatini PLL ile ayarladığımızı belirtiyoruz.

Bits 1:0 **SW:** System clock switch

Set and cleared by software to select the system clock source.

Set by hardware to force the HSI selection when leaving the Stop or Standby mode or in case of failure of the HSE oscillator used directly or indirectly as the system clock.

00: HSI oscillator selected as system clock

01: HSE oscillator selected as system clock

10: PLL selected as system clock

11: not allowed

```
RCC->CFGREG &= ~(1 << 0);
```

```
RCC->CFGREG |= (1 << 1);
```

Bits 3:2 **SWS:** System clock switch status

Set and cleared by hardware to indicate which clock source is used as the system clock.

00: HSI oscillator used as the system clock

01: HSE oscillator used as the system clock

10: PLL used as the system clock

11: not applicable

```
while(!(RCC->CR & (1 << 1)));
```

```

7 @void RCC_Config(void)
8 {
9     //RCC->CR &= 0x00000083;           //RESET
10
11    RCC->CR &= ~(1 << 0);          //HSION
12    RCC->CR |= 1 << 16;            //HSEON
13    while(!(RCC->CR & (1 << 17))); //HSERDY
14    RCC->CR |= 1 << 19;            //CSSON
15    RCC->CFGR = 0x00000000;
16    RCC->PLLCFGR |= (1 << 22);    //PLLCSR
17    //RCC->PLLCFGR &= ~(1 << 0); //PLLMO 0
18    //RCC->PLLCFGR &= ~(1 << 1); //PLLMI 0
19    //RCC->PLLCFGR |= (1 << 2); //PLLMO 1
20    //RCC->PLLCFGR &= ~(1 << 3); //PLLMI 1
21    //RCC->PLLCFGR &= ~(1 << 4); //PLLMO 2
22    //RCC->PLLCFGR &= ~(1 << 5); //PLLMI 2
23    RCC->PLLCFGR |= (4 << 0);    //PLLMO 4
24    RCC->PLLCFGR |= (168 << 6);  //PLLMI 168
25    //RCC->PLLCFGR &= ~(1 << 16); //PLLPO 0
26    //RCC->PLLCFGR &= ~(1 << 17); //PLLPI 0
27
28    RCC->CR |= 1 << 24;           //PLLON
29    while(!(RCC->CR & (1 << 25))); //PLLRDY
30
31    RCC->CFG &= ~(1 << 0);
32    RCC->CFG |= (1 << 1);        //SW
33
34    while(!(RCC->CR & (1 << 1))); //SWS
35 }
36 @int main(void)
37 {
38     //systemClock=SystemCoreClock;   //168 000 000
39
40     //RCC_DeInit();                //HSI ON PLL OFF
41
42     //SystemCoreClockUpdate();
43     //systemClock=SystemCoreClock;   //16 000 000
44
45     RCC_Config();
46     SystemCoreClockUpdate();
47     systemClock=SystemCoreClock;   //168 000 000
48
49     while (1)
50     {
51
52     }
53 }
54 }
```

Expression	Type	Value
(x)= systemClock	uint32_t	168000000

## GPIO port mode register (GPIOx\_MODER) ( $x = A..I/J/K$ )

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

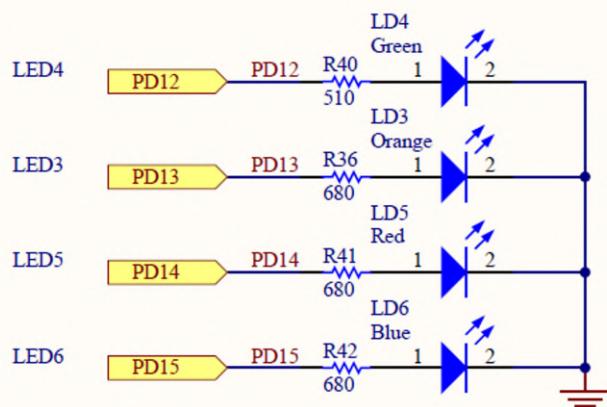
- Pinlerin nasıl kullanacağımızı seçtiğimiz kısımdır. Her iki bit bir pini temsil ediyor.
- Pini giriş, çıkış, analog ya da alternatif fonksiyon olarak mı kullanacağımızı belirtiyoruz. Alternatif fonksiyon ile kast edilen pinin çevresel birimlerden I2C, SPI olarak kullanılacağını belirtiyor.

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits ( $y = 0..15$ )

These bits are written by software to configure the I/O direction mode.

- 00: Input (reset state)
- 01: General purpose output mode
- 10: Alternate function mode
- 11: Analog mode

- STM32f407VG mikrodenetleyicisinin board üzerindeki led pinleri D12, D13, D14 ve D15'tir.



### LEDs

- D portundaki belirlediğimiz pinlerimizi output yani çıkış olarak tanımlıyoruz ama öncesinde D portunu clock hattına aktarmamız gerekiyor.

```

GPIOD->MODER |= 1 << 24;           //PD12
GPIOD->MODER &= ~(1 << 25);
GPIOD->MODER |= 1 << 26;           //PD13
GPIOD->MODER &= ~(1 << 27);
GPIOD->MODER |= 1 << 28;           //PD14
GPIOD->MODER &= ~(1 << 29);
GPIOD->MODER |= 1 << 30;           //PD15
GPIOD->MODER &= ~(1 << 31);
    
```

- Output ayarı için her bir pine bir bit ayrılmıştır. Kullanacağımız push-pull reset durumunda 0 olduğundan bir ayar yapmamıza gerek yok.

## GPIO port output type register (GPIOx\_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 OTy: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

- Pinlerin very high speed olmasını istiyoruz. Pinleri 1 yapıyoruz.

## GPIO port output speed register (GPIOx\_OSPEEDR) (x = A..I/J/K)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 OSPEEDRy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus  $V_{DD}$  range and external load.

```

GPIOD->OSPEEDR |= 1 << 24;
GPIOD->OSPEEDR |= 1 << 25;
GPIOD->OSPEEDR |= 1 << 26;
GPIOD->OSPEEDR |= 1 << 27;
GPIOD->OSPEEDR |= 1 << 28;
GPIOD->OSPEEDR |= 1 << 29;
GPIOD->OSPEEDR |= 1 << 30;
GPIOD->OSPEEDR |= 1 << 31;

```

- Bu şekilde tek tek yazmak yerine 32 bitinin son sekize 1 yazarak hex kısmı yazabiliriz.

HEX	FF00 0000
DEC	-16.777.216
OCT	37 700 000 000
BIN	1111 1111 0000 0000 0000 0000 0000 0000
	WORD MS M <sup>+</sup>
0000 0000 0000 0000	
60 56 52 48	
0000 0000 0000 0000	
44 40 36 32	
1111 1111 0000 0000	
28 24 20 16	
0000 0000 0000 0000	
12 8 4 0	

GPIOD->OSPEEDR |= 0xFF000000;

- 8 biti 1 yaptığımızda HEX olarak FF veriri. 24.bite eklemeli şekilde de yapabiliriz.

GPIOD->OSPEEDR |= FF << 24;

- Kullanacağımız no pull-up, no pull-down reset durumunda 0 olduğundan bir ayar yapmamıza gerek yok.

## GPIO port pull-up/pull-down register (GPIOx\_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]	PUPDR14[1:0]	PUPDR13[1:0]	PUPDR12[1:0]	PUPDR11[1:0]	PUPDR10[1:0]	PUPDR9[1:0]	PUPDR8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 PUPDR<sub>y</sub>[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

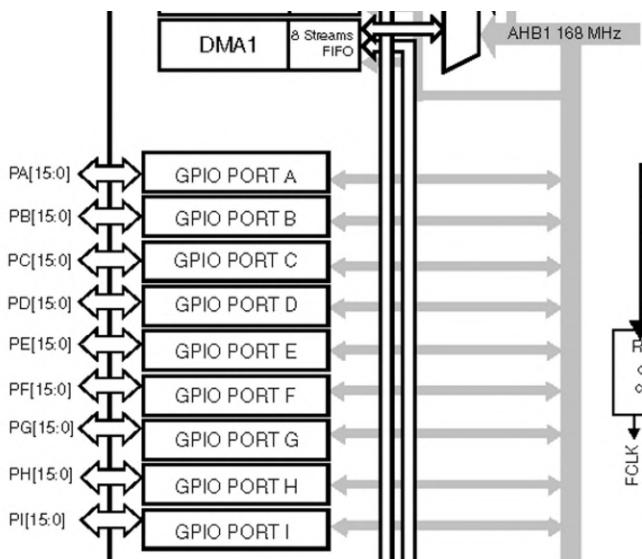
00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

- Öncesinde clock hattını aktif etmemiz gerekiyor. Portlar AHB1 kısmına gidiyor.



## RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser- ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Reserved		DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved		
	rw	rw	rw	rw	rw	rw			rw	rw			rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Reserved		CRCE N	Reserved			GPIOE N	GPIOH EN	GPIOG EN	GPIOFE N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN	
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	

- Biz D portunu kullandığımızdan sadece bunu aktif ediyoruz.

Bit 3 **GPIODEN**: IO port D clock enable

Set and cleared by software.

0: IO port D clock disabled

1: IO port D clock enabled

RCC->AHB1ENR |= (1 << 3);

- Pinleri set ya da reset edeceğimiz kısımdır. Bunları while döngüsü içerisinde yazıyoruz.

## GPIO port output data register (GPIOx\_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx\_BSRR register (x = A..I/J/K).

- ```

GPIOD->ODR |= 1 << 12;
GPIOD->ODR |= 1 << 13;
GPIOD->ODR |= 1 << 14;
GPIOD->ODR |= 1 << 15;

• Clock ayarlarını aşağıdaki gibi düzelttik.
• Önceki yaptıklarımızın hepsini HEX formatında olacak şekilde düzelttik.
• Konunun videosu eski olduğundan önceki yaptığımız örneklerden farklılık olarak eskilerde RCC_CFGR register kısmın 4:7 arası bitler koda eklenmişken yenilerde ekli olan 0:3 arası bitler eklenmemiş.
  Eski videoda döngüde açılan flagların kapatılması için RCC_CIR registerin 11. ve 23.bitleri resetlemiş.

```

```

8 void RCC_Config(void)
9 {
10    RCC->CR |= 0x00030000;           //HSEON, HSERDY
11    while(!(RCC->CR & 0x00020000)); //HSERDY
12    RCC->CR |= 0x00080000;         //CSSON
13    RCC->CFGR = 0x00000000;
14    RCC->PLLCFGR |= 0x00400000;   //PLLSRC
15    RCC->PLLCFGR |= 0x00000004;   //PLLM 4
16    RCC->PLLCFGR |= 0x00002A00;   //PLLN 168
17    RCC->PLLCFGR |= 0x00000000;   //PLLP 2
18    RCC->CR |= 0x01000000;        //PLLON
19    while(!(RCC->CR & 0x02000000)); //PLLRDY
20    RCC->CFGR |= 0x00000001;      //SW
21    while(!(RCC->CR & 0x00000001)); //SWS
22 }

• GPIO ayarlarını aşağıdaki gibi düzelttik.

19 void GPIO_Config(void)
20 {
21    RCC->AHB1ENR |= (1 << 3);     //D clock enable
22
23    //output mode
24    GPIOD->MODER |= 0x55000000;    //PD12, PD13, PD14, PD15
25
26    //Very high speed
27    GPIOD->OSPEEDR |= 0xFF000000;
28 }

```

### Kod Kısmı

- RCC ve GPIO için yazdığımız fonksiyonları ekledik ardından while dönüsü içerisinde led blink uygulamasını gerçekleştirdik.

```
36 int main(void)
37 {
38
39     RCC_Config();
40     SystemCoreClockUpdate();
41
42     GPIO_Config();
43
44     while (1)
45     {
46         //set
47         GPIOD->ODR |= 1 << 12;
48         GPIOD->ODR |= 1 << 13;
49         GPIOD->ODR |= 1 << 14;
50         GPIOD->ODR |= 1 << 15;
51
52         for(int i=0; i<1680000; i++);
53
54         //reset
55         GPIOD->ODR &= ~(1 << 12);
56         GPIOD->ODR &= ~(1 << 13);
57         GPIOD->ODR &= ~(1 << 14);
58         GPIOD->ODR &= ~(1 << 15);
59
60         for(int i=0; i<1680000; i++);
61     }
62 }
```

# Buton ile Led Yakma

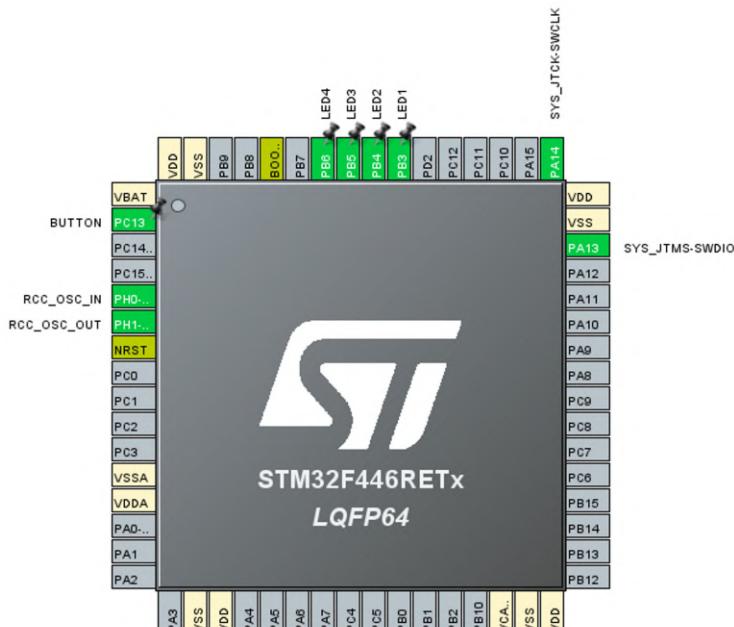
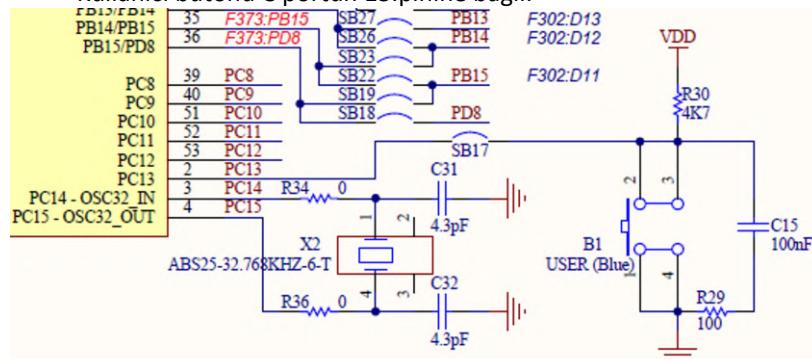
25 Aralık 2021 Cumartesi 00:52

## Buton ile Led Yakma



### Konfigürasyon Kısımları

- Kullanıcı butonu C portun 13.pinine bağlı.



| Pin Name | Signal on Pin | GPIO output... | GPIO mode      | GPIO Pull-u...  | Maximum o... | User Label | Modified |
|----------|---------------|----------------|----------------|-----------------|--------------|------------|----------|
| PB3      | n/a           | Low            | Output Push... | No pull-up a... | Very High    | LED1       | ✓        |
| PB4      | n/a           | Low            | Output Push... | No pull-up a... | Very High    | LED2       | ✓        |
| PB5      | n/a           | Low            | Output Push... | No pull-up a... | Very High    | LED3       | ✓        |
| PB6      | n/a           | Low            | Output Push... | No pull-up a... | Very High    | LED4       | ✓        |
| PC13     | n/a           | n/a            | Input mode     | Pull-down       | n/a          | BUTTON     | ✓        |

### Kod Kısımları

- RCC için bir değişiklik yapmadığımızdan fonksiyon içeriği aynıdır.
- hal\_gpio.c kısmından ReadPin fonksiyonlarına ulaşabiliriz.

```

375 GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
376 {
377     GPIO_PinState bitstatus;
378
379     /* Check the parameters */
380     assert_param(IS_GPIO_PIN(GPIO_Pin));
381
382     if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
383     {
384         bitstatus = GPIO_PIN_SET;
385     }
386     else
387     {
388         bitstatus = GPIO_PIN_RESET;
389     }
390
391     return bitstatus;
391 }

```

- Buton için count değişkeni atıyoruz.

```

44 /* USER CODE BEGIN PV */
45 int count=0;
46 /* USER CODE END PV */

```

- Okuma yaptığında yani butona bastığımızda if yapısının içine girer ve while ile kullanıcının eli butona basılı olup olmadığını kontrol ederiz. Elini çektiğinde count değerini 1 arttırıyor.
- Count değerinin modunu aldığımızda 0 iken RESET, 1 iken SET durumundadır.

```

94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         /* USER CODE BEGIN 3 */
99         if(HAL_GPIO_ReadPin(GPIOC, BUTTON_Pin))
100         {
101             while(HAL_GPIO_ReadPin(GPIOC, BUTTON_Pin));
102             HAL_Delay(100);
103             count++;
104         }
105
106         if(count % 2 == 1)
107         {
108             HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin,GPIO_PIN_SET);
109         }
110         else
111         {
112             HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin,GPIO_PIN_RESET);
113         }
114     }

```

## ➤ REGISTER

### Konfigürasyon Kısımlı

- Öncelikle RCC ve GPIO ayarlarını yapıyoruz.

```

8 void RCC_Config(void)
9 {
10     RCC->CR |= 0x00030000;           //HSEON, HSERDY
11     while(!(RCC->CR & 0x00020000)); //HSERDY
12     RCC->CR |= 0x00080000;          //CSSON
13     RCC->CFGR = 0x00000000;
14     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
15     RCC->PLLCFGR |= 0x00000004;    //PLLM 4
16     RCC->PLLCFGR |= 0x00002A00;    //PLLN 168
17     RCC->PLLCFGR |= 0x00000000;    //PLLP 2
18     RCC->CR |= 0x01000000;          //PLLON
19     while(!(RCC->CR & 0x02000000)); //PLLRDY
20     RCC->CFGR |= 0x00000001;        //SW
21     while(!(RCC->CR & 0x00000001)); //SWS
22 }

```

- GPIO ayarlarını aşağıdaki gibi düzelttik.
- A portunu da kullanacağımızdan A ve D portlarını RCC clocklarını aktif ediyoruz.

```
RCC->AHB1ENR |= 0x00000009;           //A, D clock enable
```

```

24 void GPIO_Config(void)
25 {
26     RCC->AHB1ENR |= 0x00000009;           //A, D clock enable
27
28     GPIOD->MODER |= 0x55000000;          //PD12, PD13, PD14, PD15
29     GPIOD->OTYPER |= 0x00000000;         //Output push-pull
30     GPIOD->OSPEEDR |= 0xFF000000;        //Very high speed
31     GPIOD->PUPDR |= 0x00000000;          //No pull-up, pull-down
32 }

```

### Kod Kısmı

- Buton için A0 pini için okuma yapacağız. Burada ilk biti kullanacağız. While döngüsü içinde bir if yapısı içinde eğer okuma yapılrsa while döngüsüne girer ve okuma yani butona basma devam ediyorsa count değişkenini 1 arttırır.
- Buradaki delay ark olayından dolayı kullandık.

## GPIO port input data register (GPIOx\_IDR) (x = A..I/J/K)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

| 31       | 30    | 29    | 28    | 27    | 26    | 25   | 24   | 23   | 22   | 21   | 20   | 19   | 18   | 17   | 16   |
|----------|-------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|------|
| Reserved |       |       |       |       |       |      |      |      |      |      |      |      |      |      |      |
| 15       | 14    | 13    | 12    | 11    | 10    | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| IDR15    | IDR14 | IDR13 | IDR12 | IDR11 | IDR10 | IDR9 | IDR8 | IDR7 | IDR6 | IDR5 | IDR4 | IDR3 | IDR2 | IDR1 | IDR0 |
| r        | r     | r     | r     | r     | r     | r    | r    | r    | r    | r    | r    | r    | r    | r    | r    |

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 IDRy: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

```

if(GPIOA->IDR & 0x00000001)
{
    while(GPIOA->IDR & 0x00000001);
    delay(1680000);

    count++;
}

```

- Count ve delay için globalde tanıttık.
- While döngüsünün içi 1 olduğu sürece çalışmaya devam eder ancak 0 olduğunda döngüden çıkış alt satıra geçer.

```

3 int count = 0;
4
5 void delay(uint32_t time)
6 {
7     while(time--);
8 }

```

- Count değişkenini kalanı 0 ise yani 2.defa basıldığında ledi söndürken kalan 1 olduğunda yani 1.defa basıldığı durumda ledi yakacaktır.

```
57 int main(void)
58 {
59     RCC_Config();
60     SystemCoreClockUpdate();
61
62     GPIO_Config();
63
64     while (1)
65     {
66         if(GPIOA->IDR & 0x00000001)
67         {
68             while(GPIOA->IDR & 0x00000001);
69             delay(1680000);
70
71             count++;
72         }
73         if(count %2 == 0)
74             GPIOD->ODR |= 0x00000000;
75         else
76             GPIOD->ODR |= 0x0000F000;
77     }
78 }
```

# External Interrupt

25 Aralık 2021 Cumartesi 00:54

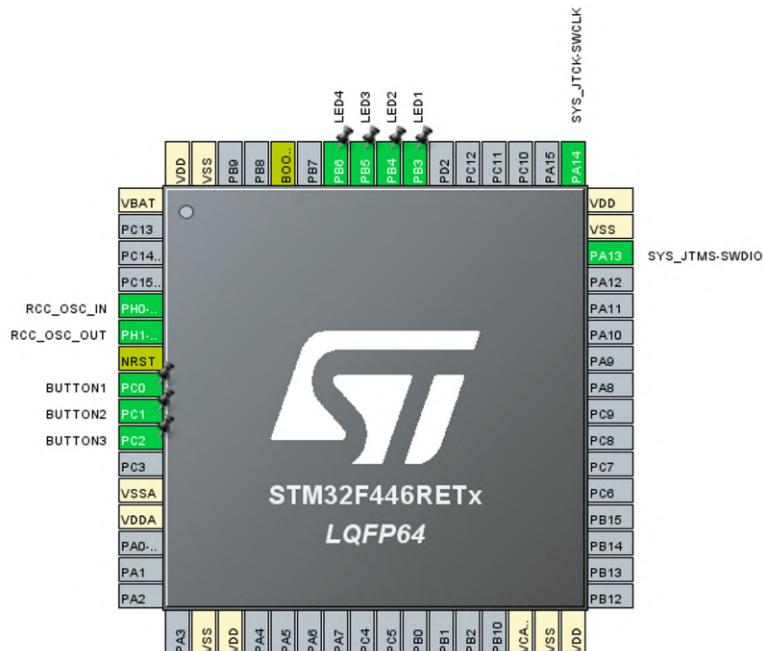
## External Interrupt

### ➤ HAL

#### Teori

- Interrupt olduğunda gittiği yerdeki fonksiyon içerisindeki kod çalışır. Buradan çıktıgı zaman kaldığı yere geri döner.
- <https://www.youtube.com/watch?v=hikauabT0bM> linkten örnek uygulamayı inceleyebiliriz.

#### Konfigürasyon Kısımları



| Pin Name | Signal on Pin | GPIO output ... | GPIO mode        | GPIO Pull-u...  | Maximum ou... | User Label | Modified |
|----------|---------------|-----------------|------------------|-----------------|---------------|------------|----------|
| PB3      | n/a           | Low             | Output Push ...  | No pull-up a... | Very High     | LED1       | ✓        |
| PB4      | n/a           | Low             | Output Push ...  | No pull-up a... | Very High     | LED2       | ✓        |
| PB5      | n/a           | Low             | Output Push ...  | No pull-up a... | Very High     | LED3       | ✓        |
| PB6      | n/a           | Low             | Output Push ...  | No pull-up a... | Very High     | LED4       | ✓        |
| PC0      | n/a           | n/a             | External Inte... | Pull-down       | n/a           | BUTTON1    | ✓        |
| PC1      | n/a           | n/a             | External Inte... | Pull-down       | n/a           | BUTTON2    | ✓        |
| PC2      | n/a           | n/a             | External Inte... | Pull-down       | n/a           | BUTTON3    | ✓        |

- NVIC kısmından işaretlediğimiz pinlerin kesme işlemi yapabilmesi için Enabled kısmı işaretliyoruz.

| NVIC Interrupt Table  | Enabled | Preemption Priority | Sub Priority |
|-----------------------|---------|---------------------|--------------|
| EXTI line 0 interrupt | ✓       | 0                   | 0            |
| EXTI line 1 interrupt | ✓       | 0                   | 0            |
| EXTI line 2 interrupt | ✓       | 0                   | 0            |

#### Kod Kısmı

- İt.c dosyasına gittiğimizde 3 adet kesme fonksiyonları oluşturulmuştur.

```
205 void EXTI0_IRQHandler(void)
206 {
207     /* USER CODE BEGIN EXTI0_IRQn_0 */
208
209     /* USER CODE END EXTI0_IRQn_0 */
210     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
211     /* USER CODE BEGIN EXTI0_IRQn_1 */
212
213     /* USER CODE END EXTI0_IRQn_1 */
214 }
```

```

219 void EXTI1_IRQHandler(void)
220 {
221     /* USER CODE BEGIN EXTI1_IRQn 0 */
222
223     /* USER CODE END EXTI1_IRQn 0 */
224     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
225     /* USER CODE BEGIN EXTI1_IRQn 1 */
226
227     /* USER CODE END EXTI1_IRQn 1 */
228 }
229
230
231 void EXTI2_IRQHandler(void)
232 {
233     /* USER CODE BEGIN EXTI2_IRQn 0 */
234
235     /* USER CODE END EXTI2_IRQn 0 */
236     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
237     /* USER CODE BEGIN EXTI2_IRQn 1 */
238
239     /* USER CODE END EXTI2_IRQn 1 */
240 }
241
242
243 void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
244 {
245     /* EXTI line interrupt detected */
246     if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
247     {
248         __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
249         HAL_GPIO_EXTI_Callback(GPIO_Pin);
250     }
251
252     • İçerisinde yazılı olana Ctrl + Space tıkladığımızda hal_gpio.c dosyaya götürür. Burada en son satırda Callback fonksiyonunu çağırır.
253
254
255     __weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
256     {
257         /* Prevent unused argument(s) compilation warning */
258         UNUSED(GPIO_Pin);
259
260         /* NOTE: This function Should not be modified, when the callback is needed,
261                 the HAL_GPIO_EXTI_Callback could be implemented in the user file
262         */
263     }
264
265     • Buna aynı şekilde gittiğimizde bu dosya üzerinde yer alır. Biz main.c dosyamızda bu fonksiyonu kullanıp kesme işlemi yapacağız.
266
267     /* Private includes -----*/
268     /* USER CODE BEGIN Includes */
269     int count=0 , i=0;
270     /* USER CODE END Includes */
271
272
273
274     while (1)
275     {
276         /* USER CODE END WHILE */
277
278         /* USER CODE BEGIN 3 */
279         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_All);
280         HAL_Delay(count);
281     }
282     /* USER CODE END 3 */
283 }
284
285     • Count değişkenini kesme olduğunda her buton için ayrı delay süreleri verdik.

```

```

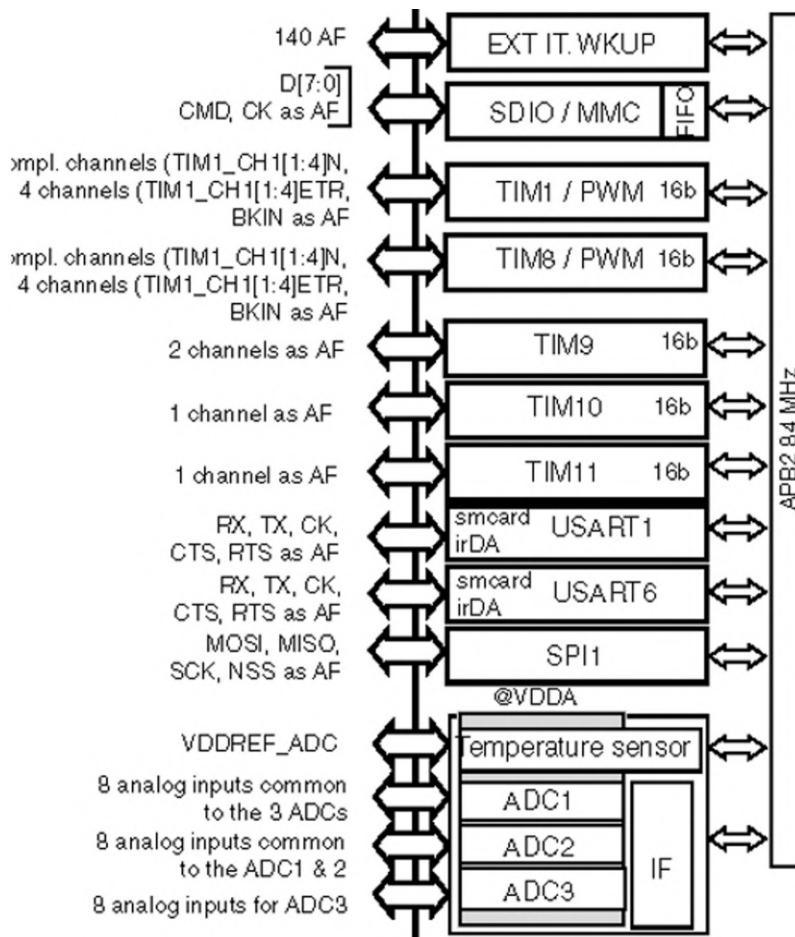
55/* Private user code -----
56 /* USER CODE BEGIN 0 */
57void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
58 {
59     i++;
60     if(HAL_GPIO_ReadPin(GPIOC, BUTTON1_Pin))
61     {
62         if(i==1)
63         {
64             count=1000;
65         }
66         else if(i==2)
67         {
68             count=750;
69         }
70         else if(i==3)
71         {
72             count=500;
73         }
74     }
75     else
76     {
77         count=250;
78         i=0;
79     }
80     else if(HAL_GPIO_ReadPin(GPIOC, BUTTON2_Pin))
81     {
82         count=100;
83     }
84     else
85     {
86         count=50;
87     }
88 }
89 /* USER CODE END 0 */

```

## ➤ REGISTER

### Konfigürasyon Kısmı

- External Interrup'in clock hattı APB2'ye gidiyor.



## RCC APB2 peripheral clock enable register (RCC\_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

|          |               |               |            |            |            |            |            |          |          |                  |                  |          |            |             |             |            |
|----------|---------------|---------------|------------|------------|------------|------------|------------|----------|----------|------------------|------------------|----------|------------|-------------|-------------|------------|
| 31       | 30            | 29            | 28         | 27         | 26         | 25         | 24         | 23       | 22       | 21               | 20               | 19       | 18         | 17          | 16          |            |
| Reserved |               |               |            |            |            |            |            |          |          |                  |                  |          |            | TIM11<br>EN | TIM10<br>EN | TIM9<br>EN |
| 15       | 14            | 13            | 12         | 11         | 10         | 9          | 8          | 7        | 6        | 5                | 4                | 3        | 2          | 1           | 0           |            |
| Reserved | SYSCFG<br>GEN | Reser-<br>ved | SPI1<br>EN | SDIO<br>EN | ADC3<br>EN | ADC2<br>EN | ADC1<br>EN | Reserved | Reserved | USART<br>6<br>EN | USART<br>1<br>EN | Reserved | Tim8<br>EN | Tim1<br>EN  |             |            |
|          | rw            |               | rw         | rw         | rw         | rw         | rw         |          |          | rw               | rw               |          | rw         | rw          |             |            |

Bit 14 **SYSCFGEN**: System configuration controller clock enable

Set and cleared by software.

0: System configuration controller clock disabled

1: System configuration controller clock enabled

- 14.pini 1 yapıyoruz.

RCC->APB2ENR |= 0x00004000; //SYSCFGEN

- 3 tane interrupt pini kullanacağımızdan öncelikle bunları aktif edip, öncelik sırası belirlemeliyiz.
- core\_cm4.h kütüphanesinde bu fonksiyonları görebiliriz.

1452<sup>⊕</sup> \_\_STATIC\_INLINE void NVIC\_EnableIRQ(IRQn\_Type IRQn)

- İkinci değere öncelik numarası yazıyoruz. Ne kadar küçük olursa öncelik sıralamasında önde olur.

1535<sup>⊕</sup> \_\_STATIC\_INLINE void NVIC\_SetPriority(IRQn\_Type IRQn, uint32\_t priority)

NVIC\_EnableIRQ(EXTI0\_IRQn);

NVIC\_EnableIRQ(EXTI1\_IRQn);

NVIC\_EnableIRQ(EXTI2\_IRQn);

```

NVIC_EnableIRQ(EXTI0_IRQn);
NVIC_EnableIRQ(EXTI1_IRQn);
NVIC_EnableIRQ(EXTI2_IRQn);

NVIC_SetPriority(EXTI0_IRQn, 0);
NVIC_SetPriority(EXTI0_IRQn, 1);
NVIC_SetPriority(EXTI0_IRQn, 2);

```

- Interrupt olarak mı event olarak mı kullanacağımızı belirliyoruz. Interrupt olarak kullanacağız.

### Interrupt mask register (EXTI\_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

| 31       | 30   | 29   | 28   | 27   | 26   | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17   | 16   |
|----------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|------|------|
| Reserved |      |      |      |      |      |     |     |     |     |     |     |     |     | MR22 | MR21 |
|          |      |      |      |      |      |     |     |     |     |     |     |     |     | rw   | rw   |
| 15       | 14   | 13   | 12   | 11   | 10   | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1    | 0    |
| MR15     | MR14 | MR13 | MR12 | MR11 | MR10 | MR9 | MR8 | MR7 | MR6 | MR5 | MR4 | MR3 | MR2 | MR1  | MR0  |
| rw       | rw   | rw   | rw   | rw   | rw   | rw  | rw  | rw  | rw  | rw  | rw  | rw  | rw  | rw   | rw   |

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **MRx**: Interrupt mask on line x

0: Interrupt request from line x is masked

1: Interrupt request from line x is not masked

- İlk 3 pine 1 yazıyoruz.

EXTI->**EMR** |= x00000007;

- Alçalan kenar mı yükselen kenar mı kullanacağımızı belirliyoruz. Yükselen kenar kullanacağız.

### Rising trigger selection register (EXTI\_RTSR)

Address offset: 0x08

Reset value: 0x0000 0000

| 31       | 30   | 29   | 28   | 27   | 26   | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17   | 16   |
|----------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|------|------|
| Reserved |      |      |      |      |      |     |     |     |     |     |     |     |     | TR22 | TR21 |
|          |      |      |      |      |      |     |     |     |     |     |     |     |     | rw   | rw   |
| 15       | 14   | 13   | 12   | 11   | 10   | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1    | 0    |
| TR15     | TR14 | TR13 | TR12 | TR11 | TR10 | TR9 | TR8 | TR7 | TR6 | TR5 | TR4 | TR3 | TR2 | TR1  | TR0  |
| rw       | rw   | rw   | rw   | rw   | rw   | rw  | rw  | rw  | rw  | rw  | rw  | rw  | rw  | rw   | rw   |

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx**: Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line

1: Rising trigger enabled (for Event and Interrupt) for input line

İlk 3 pine 1 yazıyoruz.

EXTI->**RTSR** |= x00000007;

- Butonları external interrupt için aktif etmemiz gerekiyor.

Buton için A portunu kullanıyoruz.

### SYSCFG external interrupt configuration register 1 (SYSCFG\_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

| 31         | 30 | 29 | 28 | 27         | 26 | 25 | 24 | 23         | 22 | 21 | 20 | 19         | 18 | 17 | 16 |
|------------|----|----|----|------------|----|----|----|------------|----|----|----|------------|----|----|----|
| Reserved   |    |    |    |            |    |    |    |            |    |    |    |            |    |    |    |
| 15         | 14 | 13 | 12 | 11         | 10 | 9  | 8  | 7          | 6  | 5  | 4  | 3          | 2  | 1  | 0  |
| EXTI3[3:0] |    |    |    | EXTI2[3:0] |    |    |    | EXTI1[3:0] |    |    |    | EXTI0[3:0] |    |    |    |
| rw         | rw | rw | rw | rw         | rw | rw | rw | rw         | rw | rw | rw | rw         | rw | rw | rw |

Bits 31:16 Reserved must be kept at reset value

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 0 to 3)

These bits are written by software to select the source input for the EXTIx external interrupt.

0000: PA[x] pin  
0001: PB[x] pin  
0010: PC[x] pin  
0011: PD[x] pin  
0100: PE[x] pin  
0101: PF[x] pin  
0110: PG[x] pin  
0111: PH[x] pin  
1000: PI[x] pin

SYSCFG->**EXTICR[0]** = 0x00000000;

- RCC, GPIO ve EXTI için yazdığımız fonksiyonlar aşağıdaki gibidir.

```
8@void RCC_Config(void)
9 {
10    RCC->CR |= 0x00030000;           //HSEON, HSERDY
11    while(!(RCC->CR & 0x00020000)); //HSERDY
12    RCC->CR |= 0x00080000;           //CSSON
13    RCC->CFGR = 0x00000000;
14    RCC->PLLCFGR |= 0x00400000;      //PLLSRC
15    RCC->PLLCFGR |= 0x00000004;      //PLLM 4
16    RCC->PLLCFGR |= 0x00002A00;      //PLLN 168
17    RCC->PLLCFGR |= 0x00000000;      //PLLP 2
18    RCC->CR |= 0x01000000;           //PLLON
19    while(!(RCC->CR & 0x02000000)); //PLLRDY
20    RCC->CFGR |= 0x00000001;          //SW
21    while !(RCC->CR & 0x00000001);   //SWS
22 }

24@void GPIO_Config(void)
25 {
26    RCC->AHB1ENR |= 0x00000009;       //A, D clock enable
27
28    GPIOD->MODER |= 0x55000000;      //PD12, PD13, PD14, PD15
29    GPIOD->OTYPER |= 0x00000000;      //Output push-pull
30    GPIOD->OSPEEDR |= 0xFF000000;     //Very high speed
31    GPIOD->PUPDR |= 0x00000000;      //No pull-up, pull-down
32 }

34@void EXTI_Config(void)
35 {
36    RCC->APB2ENR |= 0x00004000;      //SYSCGFEN
37
38    NVIC_EnableIRQ(EXTI0_IRQn);
39    NVIC_EnableIRQ(EXTI1_IRQn);
40    NVIC_EnableIRQ(EXTI2_IRQn);
41
42    NVIC_SetPriority(EXTI0_IRQn, 0);
43    NVIC_SetPriority(EXTI0_IRQn, 1);
44    NVIC_SetPriority(EXTI0_IRQn, 2);
45
46    EXTI->EMR |= 0x00000007;
47    EXTI->RTSR |= 0x00000007;
48
49    SYSCFG->EXTICR[0] = 0x00000000;
50 }
```

### Kod Kısımları

- İlk önce while döngüsünde tüm ledleri açıyoruz.

```

82 int main(void)
83 {
84     RCC_Config();
85     SystemCoreClockUpdate();
86     GPIO_Config();
87     EXTI_Config();
88
89     while (1)
90     {
91         GPIOD->ODR |= 0x0000F000;
92     }
93 }

```

- Pinde kesme olup olmadığını öğrenmemiz gerekiyor. Bitte değer olduğunda kesmeye girmiş oluyor. Tekrar 1 yazarak bayrağı indiriyoruz.

## Pending register (EXTI\_PR)

Address offset: 0x14

Reset value: undefined

|          |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 31       | 30    | 29    | 28    | 27    | 26    | 25    | 24    | 23    | 22    | 21    | 20    | 19    | 18    | 17    | 16    |
| Reserved |       |       |       |       |       |       |       |       |       |       |       |       |       | PR22  | PR21  |
|          |       |       |       |       |       |       |       |       | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |
| 15       | 14    | 13    | 12    | 11    | 10    | 9     | 8     | 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     |
| PR15     | PR14  | PR13  | PR12  | PR11  | PR10  | PR9   | PR8   | PR7   | PR6   | PR5   | PR4   | PR3   | PR2   | PR1   | PR0   |
| rc_w1    | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 PRx: Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line.

This bit is cleared by programming it to '1'.

- Eğer kesmeye giriliyorsa 2 saniye boyunca belirlediğimiz ledi açıp diğerlerini kapatıyoruz.

```

51 void EXTI IRQHandler()
52 {
53     if(EXTI->PR & 0x00000001)
54     {
55         GPIOD->ODR |= 0x00001000;
56         delay(33600000);
57     }
58     EXTI->PR = 0x00000001;
59 }

```

```

61 void EXTI IRQHandler()
62 {
63     if(EXTI->PR & 0x00000002)
64     {
65         GPIOD->ODR |= 0x00002000;
66         delay(33600000);
67     }
68     EXTI->PR = 0x00000002;
69 }

```

```

71 void EXTI IRQHandler()
72 {
73     if(EXTI->PR & 0x00000004)
74     {
75         GPIOD->ODR |= 0x00004000;
76         delay(33600000);
77     }
78     EXTI->PR = 0x00000004;
79 }

```

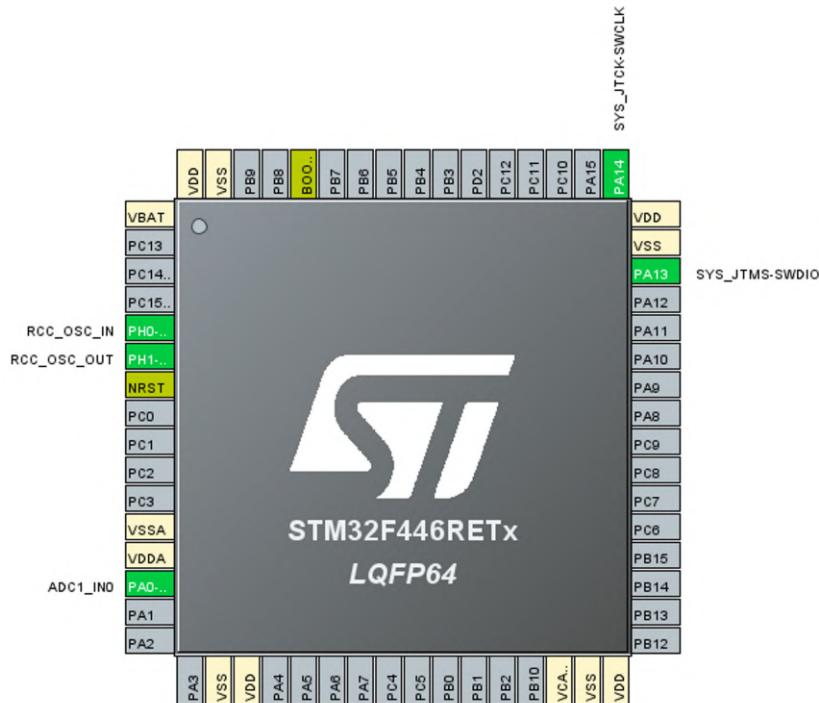
## ADC Verisi Okuma

25 Aralık 2021 Cumartesi 00:56

## ADC Verisi Okuma

➤ HAL

## Konfigürasyon Kısımları



| Pin Name | Signal on Pin | GPIO output | GPIO mode   | GPIO Pull-up/...  | Maximum... | User L... | Modified                 |
|----------|---------------|-------------|-------------|-------------------|------------|-----------|--------------------------|
| PA0-WKUP | ADC1_IN0      | n/a         | Analog mode | No pull-up and... | n/a        |           | <input type="checkbox"/> |

- Analag değer okumak için potansiyometreyi PA0'a bağladık. Bunun için Analog kısmından ADC1'den IN0 tıklarız. Gerçek voltaj değerini de hesaplayacağımızdan Vrefinit Channel'da seçilir.

- IN0
- IN1
- IN2
- IN3
- IN4
- IN5
- IN6
- IN7
- IN8
- IN9
- IN10
- IN11
- IN12
- IN13
- IN14
- IN15

- Vrefint Channel**

  - Daha sonra Parameter Settings'den Continuous Conversion Mode Enabled yapılır. Sürekli çevrim modu anlamına gelir ve tekrar tekrar okuma durumu yapar eğer çalışmazsa bir kere okur daha sonra değer okumaz.
  - Çözünürlüğümüzü 12 bit yaptık yani en fazla 4095 değerini görebiliriz
  - Çok kanal olduğundan Scan Mode Enabled yapılır.

### ADC\_Settings

- |                               |                                               |
|-------------------------------|-----------------------------------------------|
| Clock Prescaler               | PCLK2 divided by 4                            |
| Resolution                    | 12 bits (15 ADC Clock cycles)                 |
| Data Alignment                | Right alignment                               |
| Scan Conversion Mode          | Enabled                                       |
| Continuous Conversion Mode    | Enabled                                       |
| Discontinuous Conversion Mode | Disabled                                      |
| DMA Continuous Requests       | Disabled                                      |
| End Of Conversion Selection   | EOC flag at the end of single channel convers |
- Number Of Conversion kısmı 2 kanal olduğundan iki yazılır ardından rank kısmından öncelik sırası verilir.
  - Sampling Time ile kaç cycle'da bir çevrim yapacağımızı belirtiyoruz.

### ADC-Regular\_ConversionMode

|                                    |                                         |
|------------------------------------|-----------------------------------------|
| Number Of Conversion               | 2                                       |
| External Trigger Conversion Source | Regular Conversion launched by software |
| External Trigger Conversion Edge   | None                                    |
| Rank                               | 1                                       |
| Channel                            | Channel 0                               |
| Sampling Time                      | 56 Cycles                               |
| Rank                               | 2                                       |
| Channel                            | Channel Vrefint                         |
| Sampling Time                      | 56 Cycles                               |

### Kod Kısmı

- ADC okumada 3 mod var. Okumada Interrupt ve DMA kullanmayacağız. Polling Mode kullanımını yapacağız. Bunun nasıl kullanıldığını öğrenmek için hal\_adc.c dosyasından bakıyoruz.

```
87 *** Polling mode IO operation ***
88 =====
89 [...]
90     (+) Start the ADC peripheral using HAL_ADC_Start()
91     (+) Wait for end of conversion using HAL_ADC_PollForConversion(), at this stage
92         user can specify the value of timeout according to his end application
93     (+) To read the ADC converted values, use the HAL_ADC_GetValue() function.
94     (+) Stop the ADC peripheral using HAL_ADC_Stop()
```

- ADC çevre birimini başlatmak için 717.satırındaki fonksiyon kullanılır.
- "\*" var ise adresleme için "&" işaretini koymamız gerekiyor.

717 HAL\_StatusTypeDef HAL\_ADC\_Start(ADC\_HandleTypeDef\* hadc)

- Çevrimin takibini 883.satırındaki fonksiyon ile yapıyoruz.
- Timeout için milisecond cinsinden istiyor.

883 HAL\_StatusTypeDef HAL\_ADC\_PollForConversion(ADC\_HandleTypeDef\* hadc, uint32\_t Timeout)

1566 uint32\_t HAL\_ADC\_GetValue(ADC\_HandleTypeDef\* hadc)

840 HAL\_StatusTypeDef HAL\_ADC\_Stop(ADC\_HandleTypeDef\* hadc)

- İşlemcinin beslemesi gerçekte 3.3V değildir. Bunun için ayrı hesap yapılması gereklidir. Hesap için 2 değer için birini ADC okumasından diğerini ise adres kısmından bulacağız.
- Adres kısmı için datasheet kısmından bakılması gereklidir.

### Internal reference voltage calibration values

| Symbol                 | Parameter                                              | Memory address            |
|------------------------|--------------------------------------------------------|---------------------------|
| V <sub>REFIN_CAL</sub> | Raw data acquired at temperature of 30 °C VDDA = 3.3 V | 0x1FFF 7A2A - 0x1FFF 7A2B |

- Bu adres için kodda tanımlama yapılması gerekiyor.
- Tanımlama yaparken bizim kullanacağımız bit 12, adres biti 32 bittir. Bizim burada 32 britten 16 bite dönüştürme yapmamız gerekiyor.

```

23/* Private includes -----
24 /* USER CODE BEGIN Includes */
25 #define VREFIN_CAL ((uint16_t*) ((uint32_t) 0x1FFF7A2A))
26
27 uint16_t adc_value[2];
28 float Vadc=0,Vdda=0;
29 int count=0;
30 /* USER CODE END Includes */
• İki kanalı adc_value değişkenine dizi halinde yazdık. 0.dizi rank'daki ayara göre ilk kanal oluyor.
• Diğer kanalı okuması için count değişkeni koyduk.
60/* Private user code -----
61 /* USER CODE BEGIN 0 */
62void Read_ADC()
63 {
64     HAL_ADC_Start(&hadc1);
65
66     if(HAL_ADC_PollForConversion(&hadc1, 100000) == HAL_OK)
67     {
68         adc_value[count] = HAL_ADC_GetValue(&hadc1);
69         count++;
70
71         if(count==2)
72         {
73             count=0;
74         }
75
76         //Vadc=3.3*adc_value/4095;
77
78         Vdda=3.3*(*VREFIN_CAL)/adc_value[1];
79         Vadc=Vdda*adc_value[0]/4095;
80     }
81     HAL_ADC_Stop(&hadc1);
82 }
83 /* USER CODE END 0 */

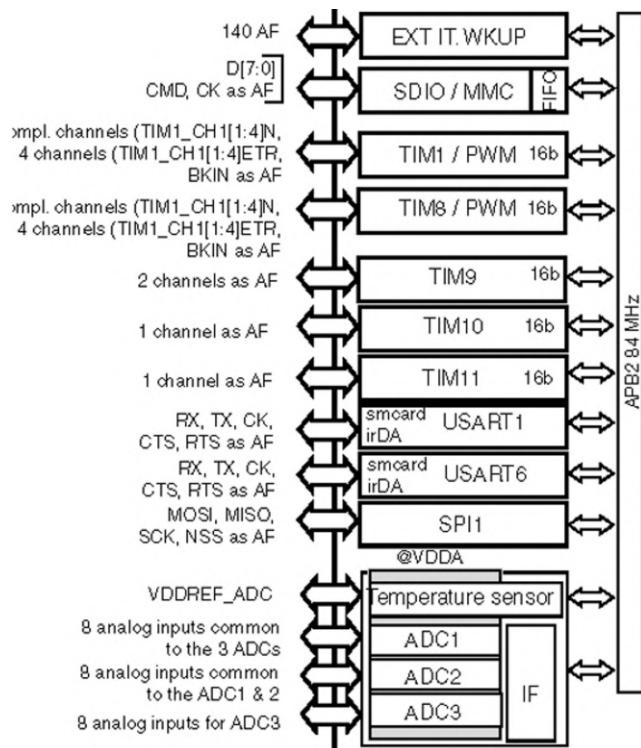
```

| Variable Name | Address/Expression | Read Value |
|---------------|--------------------|------------|
| Vadc          | 0x20000028         | 1.7121019  |
| Vdda          | 0x2000002c         | 2.9433491  |
| count         | 0x20000030         | 0          |

## ➤ REGISTER

### Konfigürasyon Kısmı

- ADC'nin clock hattı APB2'ye gidiyor.



## RCC APB2 peripheral clock enable register (RCC\_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

Bit 14 **SYSCFGEN**: System configuration controller clock enable

Set and cleared by software.

0: System configuration controller clock disabled

### 1: System configuration controller clock enabled

- 8.pini 1 yapıyoruz.

```
RCC->APB2ENR |= 0x00000100; //ADC1
```

## ADC control register 1 (ADC\_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

Bits 25:24 **RES[1:0]**: Resolution

These bits are written by software to select the resolution of the conversion.

- 00: 12-bit (15 ADCCLK cycles)
  - 01: 10-bit (13 ADCCLK cycles)
  - 10: 8-bit (11 ADCCLK cycles)
  - 11: 6-bit (9 ADCCLK cycles)

- 8 bit çözünürlük ayarlıyoruz.

```
ADC1->CR1 |= 0x02000000; //RES 8-bit
```

## ADC control register 2 (ADC\_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

Bit 0 **ADON**: A/D Converter ON / OFF

This bit is set and cleared by software.

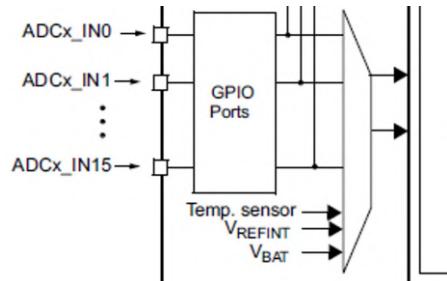
Note: 0: Disable ADC conversion and go to power down mode

## 1: Enable ADC

- ADC aktif etmek için ilk bit 1 yapıldı.

```
ADC1->CR2 |= 0x00000001; //ADON
```

- ADC\_SPMR register kısmında kaç cycle'da bir çevrim yapacağımızı belirtiyoruz.
  - 0'dan 18'e kadar kanal var.



## ADC sample time register 1 (ADC\_SMPR1)

Address offset: 0x0C

Reset value: 0x0000 0000

## ADC sample time register 2 (ADC\_SMPR2)

Address offset: 0x10

Reset value: 0x0000 0000

- Biz 0.pinde çalıştığımız için 0.kanal kullanıyoruz.
- Bits 31:30 Reserved, must be kept at reset value.

Bits 29:0 **SMPx[2:0]**: Channel x sampling time selection

These bits are written by software to select the sampling time individually for each channel.

During sample cycles, the channel selection bits must remain unchanged.

Note: 000: 3 cycles  
 001: 15 cycles  
 010: 28 cycles  
 011: 56 cycles  
 100: 84 cycles  
 101: 112 cycles  
 110: 144 cycles  
 111: 480 cycles

- 56 cycles'da çalışacağız.

`ADC1->SMPR2 |= 0x00000003;` //SMP0 56 cycles

## ADC common control register (ADC\_CCR)

Address offset: 0x04 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

|          |     |      |            |    |    |    |          |         |       |          |            |    |    |        |    |
|----------|-----|------|------------|----|----|----|----------|---------|-------|----------|------------|----|----|--------|----|
| 31       | 30  | 29   | 28         | 27 | 26 | 25 | 24       | 23      | 22    | 21       | 20         | 19 | 18 | 17     | 16 |
| Reserved |     |      |            |    |    |    |          | TSVREFE | VBATE | Reserved |            |    |    | ADCPRE |    |
| 15       | 14  | 13   | 12         | 11 | 10 | 9  | 8        | 7       | 6     | 5        | 4          | 3  | 2  | 1      | 0  |
| DMA[1:0] | DDS | Res. | DELAY[3:0] |    |    |    | Reserved |         |       |          | MULTI[4:0] |    |    |        |    |
| rw       | rw  | rw   | rw         | rw | rw | rw | rw       | rw      | rw    | rw       | rw         | rw | rw | rw     | rw |

Bits 17:16 **ADCPRE**: ADC prescalers

Set and cleared by software to select the frequency of the clock to the ADC. The clock is common for all the ADCs.

Note: 00: PCLK2 divided by 2  
 01: PCLK2 divided by 4  
 10: PCLK2 divided by 6  
 11: PCLK2 divided by 8

- ADC birimin maksimum clock hızı 36 MHz olması gerekiğinden CubeMX programında baktığımızda ADC birimin bağlı olduğu APB2 hattı 84MHz olduğundan bu clock hızını 4'e böldüğümüzde sağlamış oluyor. Bu sebeple 16. ve 17. bitleri ona göre düzenliyoruz.

`ADC->CCR |= 0x00010000;`

- RCC, GPIO ve ADC için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

21 void GPIO_Config(void)
22 {
23     RCC->AHB1ENR |= 0x00000001;           //A clock enable
24
25     GPIOA->MODER |= 0x00000003;          //PA0 Analog mode
26     GPIOA->OSPEEDR |= 0xFF000000;        //Very high speed
27 }

5 void RCC_Config(void)
6 {
7     RCC->CR |= 0x00030000;                //HSEON, HSERDY
8     while(!(RCC->CR & 0x00020000));      //HSERDY
9     RCC->CR |= 0x00080000;                //CSSON
10    RCC->CFGR = 0x00000000;
11    RCC->PLLCFGR |= 0x00400000;          //PLLCSR
12    RCC->PLLCFGR |= 0x00000004;          //PLLM 4
13    RCC->PLLCFGR |= 0x00002A00;          //PLLN 168
14    RCC->PLLCFGR |= 0x00000000;          //PLLP 2
15    RCC->CR |= 0x01000000;                //PLLON
16    while(!(RCC->CR & 0x02000000));      //PLLRDY
17    RCC->CFGR |= 0x00000001;              //SW
18    while(!(RCC->CR & 0x00000001));      //SWS
19 }

```

`20 void ADC_Config(void)`

```

29 void ADC_Config(void)
30 {
31     RCC->APB2ENR |= 0x00000100;           //ADC1
32
33     ADC1->CR1 |= 0x02000000;           //RES 8-bit
34     ADC1->CR2 |= 0x00000001;           //ADON
35     ADC1->SMPR2 |= 0x00000003;         //SMP0 56 cycles
36     ADC->CCR |= 0x00010000;           //ADCPRE 4
37 }

```

### Kod Kısmı

- 8 bitlik değişken okuyacağımızdan ona göre değişken ataması yapıyoruz.

```
3 uint8_t adc_value;
```

- 8 bitlik dönüşüm değeri olan bir ADC okuma fonksiyonu yazıyoruz ve bunu while döngüsünde adc\_value değerini Read\_ADC() fonksiyonunda tutmak için eşitliyoruz.
- Fonksiyonun içine değişken ataması yaptık.

```
40 uint8_t Read_ADC()
```

```
41 {
```

```
42     uint8_t value;
```

```
43 }
```

```
adc_value = Read_ADC();
```

## ADC control register 2 (ADC\_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

|          | 31       | 30    | 29    | 28          | 27  | 26  | 25       | 24        | 23     | 22 | 21           | 20 | 19 | 18       | 17   | 16   |
|----------|----------|-------|-------|-------------|-----|-----|----------|-----------|--------|----|--------------|----|----|----------|------|------|
| reserved | SWST ART | EXTEN |       | EXTSEL[3:0] |     |     | reserved | JSWST ART | JEXTEN |    | JEXTSEL[3:0] |    |    | reserved | CONT | ADON |
|          | rw       | rw    | rw    | rw          | rw  | rw  |          | rw        | rw     | rw | rw           | rw | rw |          |      |      |
| 15       | 14       | 13    | 12    | 11          | 10  | 9   | 8        | 7         | 6      | 5  | 4            | 3  | 2  | 1        | 0    |      |
| reserved |          |       | ALIGN | EOCS        | DDS | DMA | Reserved |           |        |    |              |    | rw | rw       | rw   | rw   |
|          |          |       | rw    | rw          | rw  | rw  |          |           |        |    |              |    |    |          |      |      |

- ADC'nin yazılımsal olarak çevrimi başlatmak için 30.biti 1 yapıyoruz.

Bit 30 **SWSTART**: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of regular channels

Note: This bit can be set only when ADON = 1 otherwise no conversion is launched.

```
ADC1->CR2 |= 0x40000000;           //SWSTART
```

- Çevrimin bitip bitmediğini ADC\_SR registerinden öğreniyoruz.

## ADC status register (ADC\_SR)

Address offset: 0x00

Reset value: 0x0000 0000

|          | 31    | 30    | 29    | 28    | 27    | 26    | 25    | 24    | 23    | 22  | 21   | 20    | 19   | 18  | 17  | 16 |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----|------|-------|------|-----|-----|----|
| Reserved |       |       |       |       |       |       |       |       |       |     |      |       |      |     |     |    |
| 15       | 14    | 13    | 12    | 11    | 10    | 9     | 8     | 7     | 6     | 5   | 4    | 3     | 2    | 1   | 0   |    |
| Reserved |       |       |       |       |       |       |       |       |       |     |      |       |      |     |     |    |
| rc_w0    | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | OVR | STRT | JSTRT | JEOC | EOC | AWD |    |

- 1.bit 1 oldu ise çevrim bittiği anlamına geliyor.

Bit 1 **EOC**: Regular channel end of conversion

This bit is set by hardware at the end of the conversion of a regular group of channels. It is cleared by software or by reading the ADC\_DR register.

0: Conversion not complete (EOCS=0), or sequence of conversions not complete (EOCS=1)

1: Conversion complete (EOCS=0), or sequence of conversions complete (EOCS=1)

```
while(!(ADC1->SR & 0x00000002));           //EOC
```

- Daha sonra ADC\_CR registerinden okuma yapıyoruz.

## ADC regular data register (ADC\_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

|            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| DATA[15:0] |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| r          | r  | r  | r  | r  | r  | r  | r  | r  | r  | r  | r  | r  | r  | r  | r  |

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 DATA[15:0]: Regular data

These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in [Figure 48](#) and [Figure 49](#).

- Okunan değeri value değişkenine atıyoruz.

```
value=ADC1->DR;
```

```
40 uint8_t Read_ADC()
41 {
42     uint8_t value;
43
44     ADC1->CR2 |= 0x40000000;           //SWSTART
45     while(!(ADC1->SR & 0x00000002));   //EOC
46
47     value=ADC1->DR;                  //DATA
48     return value;
49 }
```

```
51 int main(void)
52 {
53     RCC_Config();
54     SystemCoreClockUpdate();
55     GPIO_Config();
56     ADC_Config();
57
58     while (1)
59     {
60         adc_value = Read_ADC();
61     }
62 }
```

- STMStudio programını açıp File kısmından Import variables tıklıyoruz ve çalışma dosyamızın debug içerisindeki elf. Uzantılı dosyayı seçiyoruz ve içerisinde okuyacağımız değişkeni import edip yandaki grafiğe ekliyoruz.
- A0'a bağladığımız potansiyometreyi çevirdikçe değer değişecektir.

| Variable Name | Address/Expression | Read Value |
|---------------|--------------------|------------|
| adc_value     | 0x200000a8         | 253        |

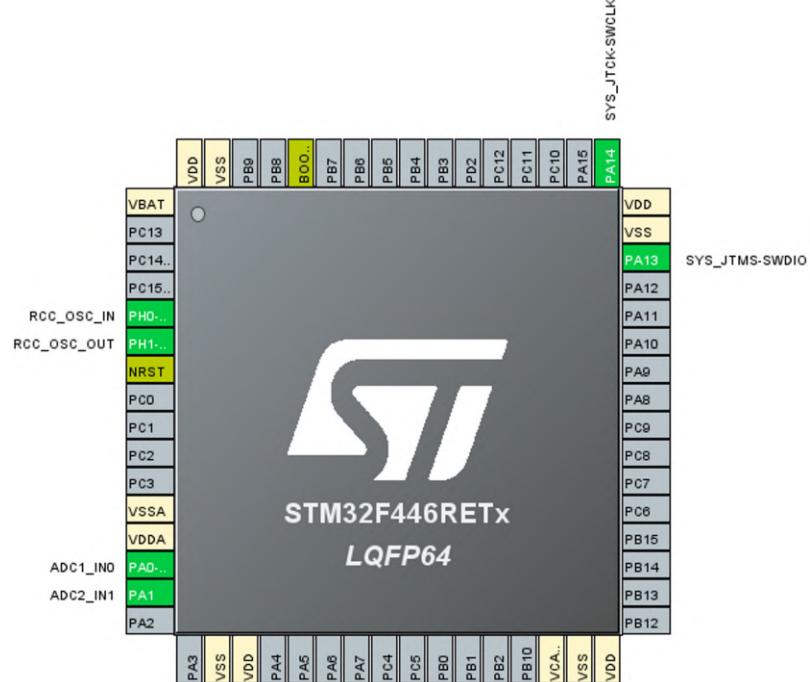
# ADC Interrupt

25 Aralık 2021 Cumartesi 00:56

## ADC Interrupt

➤ HAL

## Konfigürasyon Kısımları



| Pin Name | Signal on Pin | GPIO output I... | GPIO mode   | GPIO Pull-up/... | Maximum out... | User Label | Modified                 |
|----------|---------------|------------------|-------------|------------------|----------------|------------|--------------------------|
| PA0-WKUP | ADC1_IN0      | n/a              | Analog mode | No pull-up an... | n/a            |            | <input type="checkbox"/> |
| PA1      | ADC2_IN1      | n/a              | Analog mode | No pull-up an... | n/a            |            | <input type="checkbox"/> |

- ADC1 için IN0, Temperature Sensor Channel ve Vrefint Channel seçimi yaptık.

✓ INO

IN1

□ IN2

□ IN3

□ IN4

IN5

IN6

11

10

IN10

□ IN11

□ IN12

□ IN13

□ IN14

IN15

Tem

Vref

- Scan Mode ile Continuous Mode Enabled yapılır.

#### ADC\_Settings

|                               |                                                  |
|-------------------------------|--------------------------------------------------|
| Clock Prescaler               | PCLK2 divided by 4                               |
| Resolution                    | 12 bits (15 ADC Clock cycles)                    |
| Data Alignment                | Right alignment                                  |
| Scan Conversion Mode          | Enabled                                          |
| Continuous Conversion Mode    | Enabled                                          |
| Discontinuous Conversion Mode | Disabled                                         |
| DMA Continuous Requests       | Disabled                                         |
| End Of Conversion Selection   | EOC flag at the end of single channel conversion |

- ADC1 için 3 kanal olduğundan her biri için rank işlemi yapılır.

#### ADC-Regular\_ConversionMode

|                                    |                                         |
|------------------------------------|-----------------------------------------|
| Number Of Conversion               | 3                                       |
| External Trigger Conversion Source | Regular Conversion launched by software |
| External Trigger Conversion Edge   | None                                    |
| Rank                               | 1                                       |
| Channel                            | Channel 0                               |
| Sampling Time                      | 56 Cycles                               |
| Rank                               | 2                                       |
| Channel                            | Channel Vrefint                         |
| Sampling Time                      | 56 Cycles                               |
| Rank                               | 3                                       |
| Channel                            | Channel Temperature Sensor              |
| Sampling Time                      | 56 Cycles                               |

- ADC2 için IN1 seçimi yapılır.
- ADC2 için tek kanal olduğundan Scan Mode seçimi yapılmaz.

#### ADC\_Settings

|                               |                                                  |
|-------------------------------|--------------------------------------------------|
| Clock Prescaler               | PCLK2 divided by 4                               |
| Resolution                    | 12 bits (15 ADC Clock cycles)                    |
| Data Alignment                | Right alignment                                  |
| Scan Conversion Mode          | Disabled                                         |
| Continuous Conversion Mode    | Enabled                                          |
| Discontinuous Conversion Mode | Disabled                                         |
| DMA Continuous Requests       | Disabled                                         |
| End Of Conversion Selection   | EOC flag at the end of single channel conversion |

#### ADC-Regular\_ConversionMode

|                                    |                                         |
|------------------------------------|-----------------------------------------|
| Number Of Conversion               | 1                                       |
| External Trigger Conversion Source | Regular Conversion launched by software |
| External Trigger Conversion Edge   | None                                    |
| Rank                               | 1                                       |
| Channel                            | Channel 1                               |
| Sampling Time                      | 56 Cycles                               |

- Interrupt kullanacağımızdan NVIC Settings kısmından Enabled yapılır.

Sadece ADC1 değil ADC2 ve ADC3 için interrupts Enabled yapılmış olur.

| NVIC Interrupt Table           | Enabled                             | Preemption Priority | Sub Priority |
|--------------------------------|-------------------------------------|---------------------|--------------|
| ADC1, ADC2 and ADC3 interrupts | <input checked="" type="checkbox"/> | 0                   | 0            |

#### Kod Kısıtları

- ADC için Interrupt Mode kullanımı yapacağız.

```
96     *** Interrupt mode IO operation ***
97     =====
98     [...]
99     (+) Start the ADC peripheral using HAL_ADC_Start_IT()
100    (+) Use HAL_ADC_IRQHandler() called under ADC_IRQHandler() Interrupt subroutine
101    (+) At ADC end of conversion HAL_ADC_ConvCpltCallback() function is executed and user can
102        add his own code by customization of function pointer HAL_ADC_ConvCpltCallback
103    (+) In case of ADC Error, HAL_ADC_ErrorCallback() function is executed and user can
104        add his own code by customization of function pointer HAL_ADC_ErrorCallback
105    (+) Stop the ADC peripheral using HAL_ADC_Stop_IT()
```

```
1038@ HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc)
```

- Keseme olduğunda it.c dosyasındaki ADC\_IRQHandler fonksiyonuna gelir ve fonksiyon içerisindeki HAL\_ADC\_IRQHandler fonksiyonunu çalıştırır.
- Bizim main.c dosyasında hazır olarak yazılmıştır.

```
206@ void ADC_IRQHandler(void)
```

```
207 {  
208     /* USER CODE BEGIN ADC_IRQHandler_0 */  
209  
210     /* USER CODE END ADC_IRQHandler_0 */  
211     HAL_ADC_IRQHandler(&hadc1);  
212     HAL_ADC_IRQHandler(&hadc2);  
213     /* USER CODE BEGIN ADC_IRQHandler_1 */  
214  
215     /* USER CODE END ADC_IRQHandler_1 */  
216 }
```

- 1578.satırındaki fonksiyonu main.c dosyasında kullanarak Interrupt'a girdiğinde çalışacak.

```
1578@ __weak void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
```

- İçerisine önce hangi ADC'yi çalıştırıldığını öğrenmemiz gerekiyor. Bunun için \_HAL\_ADC\_GET\_FLAG kullanmamız gerekiyor.

```
129 (+) __HAL_ADC_GET_FLAG: Get the selected ADC's flag status
```

```
548 #define __HAL_ADC_GET_FLAG(__HANDLE__, __FLAG__) (((__HANDLE__)->Instance->SR) & (__FLAG__)) == (__FLAG__)  
• Temperature için Datasheet kısmından aşağıdaki tabloları kullanıyoruz.
```

Temperature sensor characteristics

| Symbol                             | Parameter                                                      | Min | Typ  | Max | Unit  |
|------------------------------------|----------------------------------------------------------------|-----|------|-----|-------|
| T <sub>L</sub> <sup>(1)</sup>      | V <sub>SENSE</sub> linearity with temperature                  | -   | ±1   | ±2  | °C    |
| Avg_Slope <sup>(1)</sup>           | Average slope                                                  | -   | 2.5  | -   | mV/°C |
| V <sub>25</sub> <sup>(1)</sup>     | Voltage at 25 °C                                               | -   | 0.76 | -   | V     |
| t <sub>START</sub> <sup>(2)</sup>  | Startup time                                                   | -   | 6    | 10  | μs    |
| T <sub>S_temp</sub> <sup>(2)</sup> | ADC sampling time when reading the temperature (1 °C accuracy) | 10  | -    | -   | μs    |

Temperature sensor calibration values

| Symbol  | Parameter                                                                   | Memory address            |
|---------|-----------------------------------------------------------------------------|---------------------------|
| TS_CAL1 | TS ADC raw data acquired at temperature of 30 °C, V <sub>DDA</sub> = 3.3 V  | 0x1FFF 7A2C - 0x1FFF 7A2D |
| TS_CAL2 | TS ADC raw data acquired at temperature of 110 °C, V <sub>DDA</sub> = 3.3 V | 0x1FFF 7A2E - 0x1FFF 7A2F |

- Aşağıdaki formül üzerinden hesaplıyoruz.

$$\text{Temperature (in } ^\circ\text{C)} = \{(V_{\text{SENSE}} - V_{25}) / \text{Avg\_Slope}\} + 25$$

- V<sub>25</sub> = V<sub>SENSE</sub> value for 25° C
- Avg\_Slope = average slope of the temperature vs. V<sub>SENSE</sub> curve (given in mV/°C or μV/°C)

```
23@ /* Private includes -----  
24 /* USER CODE BEGIN Includes */  
25 #define VREFIN_CAL ((uint16_t*)((uint32_t)0x1FFF7A2A))  
26 #define V25 (float) 0.76  
27 #define Avg_Slope (float) 0.0025  
28  
29 uint16_t adc1_value[3], adc2_value;  
30 float Vadc1, Vadc2, Vsense, Vdda, temperature;  
31 int count=0;  
32 /* USER CODE END Includes */
```

```
121 /* USER CODE BEGIN 2 */  
122 HAL_ADC_Start_IT(&hadc1);  
123 HAL_ADC_Start_IT(&hadc2);  
124 /* USER CODE END 2 */
```

```

65/* Private user code -----*/
66 /* USER CODE BEGIN 0 */
67 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
68 {
69     if(__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_EOC) != RESET)
70     {
71         adc1_value[count]= HAL_ADC_GetValue(&hadc1);
72         count++; // count=0 => IN0, count=1 => Vrefint, count=2 => Temperature
73         if(count==3)
74         {
75             count=0;
76         }
77         Vdda= (float) 3.3 * (*VREFIN_CAL) / adc1_value[1];
78         Vadc1= Vdda * adc1_value[0] / 4095;
79
80         Vsense = Vdda * adc1_value[2] / 4095;
81         temperature= ((Vsense - V25) / Avg_Slope) + 25;
82     }
83     if(__HAL_ADC_GET_FLAG(&hadc2,ADC_FLAG_EOC) != RESET)
84     {
85         adc2_value=HAL_ADC_GetValue(&hadc2);
86         Vadc2= Vdda * adc2_value / 4095;
87     }
88 }
89 /* USER CODE END 0 */

```

| Variable Name | Address/Expression | Read Value |
|---------------|--------------------|------------|
| Vadc1         | 0x20000028         | 2.9433491  |
| Vadc2         | 0x2000002c         | 1.9205443  |
| Vdda          | 0x20000030         | 2.9363744  |
| Vsense        | 0x20000034         | 0.782018   |
| count         | 0x20000038         | 0          |
| temperature   | 0x2000003c         | 33.065987  |

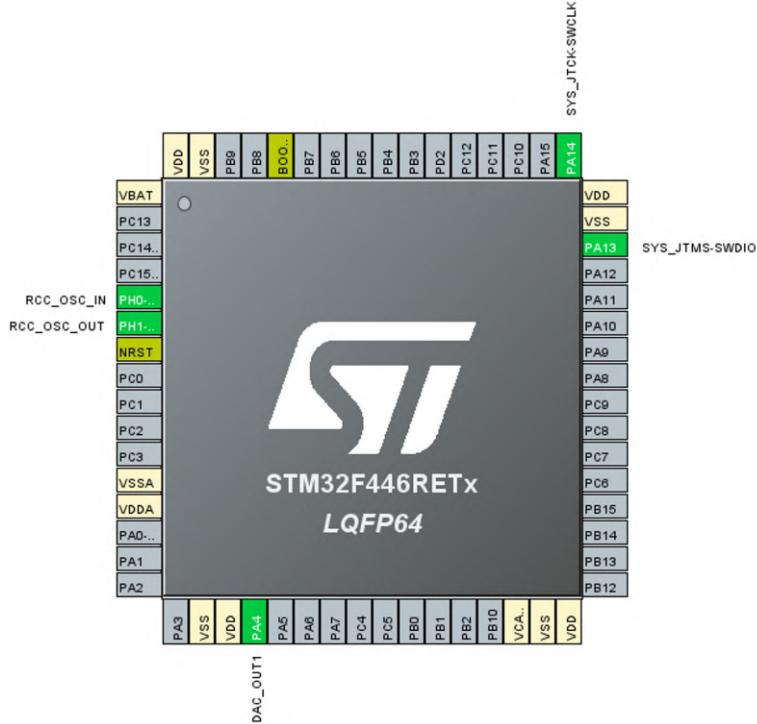
# DAC Kullanımı

25 Aralık 2021 Cumartesi 00:57

## DAC Kullanımı

### ➤ HAL

#### Konfigürasyon Kısmı



| Pin N... | Signal on... | GPIO out... | GPIO mo... | GPIO Pul...  | Maximu... | User Label | Modified                 |
|----------|--------------|-------------|------------|--------------|-----------|------------|--------------------------|
| PA4      | DAC_O...     | n/a         | Analog ... | No pull-u... | n/a       |            | <input type="checkbox"/> |

- DAC kısmından OUT1 seçiyoruz.
- OUT1 Configuration
- OUT2 Configuration
- External Trigger
- Output Buffer, gürültü engelleme; Trigger tetiklemedir.  
Biz bunları aynı şekilde bırakıyoruz.

#### DAC Out1 Settings

|               |        |
|---------------|--------|
| Output Buffer | Enable |
| Trigger       | None   |

#### Kod Kısmı

- DAC için toplam iki kanal var.
- ```
18     [...]
19     *** DAC Channels ***
20     =====
21     [...]
22     STM32F4 devices integrate two 12-bit Digital Analog Converters
23
24     The 2 converters (i.e. channel1 & channel2)
25     can be used independently or simultaneously (dual mode):
26         (#) DAC channel1 with DAC_OUT1 (PA4) as output
27         (#) DAC channel2 with DAC_OUT2 (PA5) as output
• DAC kullanmak için iki modumuz var. Biri Polling diğeri DMA'dır.
• Biz Polling Mode kullanıyoruz.
```

```

110     *** Polling mode IO operation ***
111     =====
112     [...]
113         (+) Start the DAC peripheral using HAL_DAC_Start()
114         (+) To read the DAC last data output value, use the HAL_DAC_GetValue() function.
115         (+) Stop the DAC peripheral using HAL_DAC_Stop()

435 HAL_StatusTypeDef HAL_DAC_Start(DAC_HandleTypeDef *hdac, uint32_t Channel)
    • DAC için değer verme yapacağımızdan GetValue değil SetValue kullanacağız. Çıkışın voltaj değeri ile oynamış oluyoruz.

759 /**
760  * @brief Set the specified data holding register value for DAC channel.
761  * @param hdac pointer to a DAC_HandleTypeDef structure that contains
762  *             the configuration information for the specified DAC.
763  * @param Channel The selected DAC channel.
764  *             This parameter can be one of the following values:
765  *             @arg DAC_CHANNEL_1: DAC Channel1 selected
766  *             @arg DAC_CHANNEL_2: DAC Channel2 selected
767  * @param Alignment Specifies the data alignment.
768  *             This parameter can be one of the following values:
769  *             @arg DAC_ALIGN_8B_R: 8bit right data alignment selected
770  *             @arg DAC_ALIGN_12B_L: 12bit left data alignment selected
771  *             @arg DAC_ALIGN_12B_R: 12bit right data alignment selected
772  * @param Data Data to be loaded in the selected data holding register.
773  * @retval HAL status
774 */
775 HAL_StatusTypeDef HAL_DAC_SetValue(DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)
    • Data kısmına 12 bit çalıştığımızdan 0-4095 arasında değer yazabiliyoruz.

23 /* Private includes -----
24 /* USER CODE BEGIN Includes */
25 int i=0;
26 /* USER CODE END Includes */

91 /* USER CODE BEGIN 2 */
92 HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
93 /* USER CODE END 2 */

95 /* Infinite loop */
96 /* USER CODE BEGIN WHILE */
97 while (1)
98 {
99     /* USER CODE END WHILE */
100
101    /* USER CODE BEGIN 3 */
102    do
103    {
104        HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, i);
105        HAL_Delay(1);
106        i++;
107    }
108    while(i<4096);
109    i=0;
110}
111/* USER CODE END 3 */

```

## ➤ REGISTER

### Konfigürasyon Kısımları

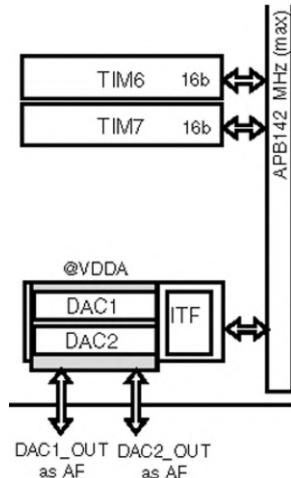
- RCC için 5.satırda 0x00030000 yerine 0x00010000 yazdık.  
HSERDY kısmı sadece okunabilir olduğundan o biti 1 yapmıyoruz.

```

3 @void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;           //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;      //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;       //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;       //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;       //PLLP 2
13    RCC->CR |= 0x01000000;           //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;          //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17 }

```

- ADC'nin clock hattı APB1'e gitiyor.



### 7.3.13 RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	RW	RW		RW	RW		RW	RW	RW	RW	RW	RW	RW	RW	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
RW	RW		RW		RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	

- 29.pini 1 yapıyoruz.

Bit 29 **DACEN**: DAC interface clock enable

Set and cleared by software.

0: DAC interface clock disabled

1: DAC interface clock enable

RCC->APB1ENR |= 0x20000000; //DACEN

## DAC control register (DAC\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

- DAC işlemi için channel1 kullanacağımızdan 0.biti aktif etmemiz gerekiyor.

Bit 0 EN1: DAC channel1 enable

This bit is set and cleared by software to enable/disable DAC channel1.

0: DAC channel1 disabled

1: DAC channel1 enabled

```
DAC->CR |= 0x00000001;
```

- Yazılımsal tetikleme kullanmadığımızdan 0. biti 0 yapıyoruz.

## DAC software trigger register (DAC\_SWTRIGR)

Address offset: 0x04

Reset value: 0x0000 0000

Bit 0 **SWTRIG1**: DAC channel1 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

0: Software trigger disabled

### 1: Software trigger enabled

**Note:** This bit is cleared by hardware (one APB1 clock cycle later) once the DAC\_DHR1 register value has been loaded into the DAC\_DOR1 register.

DAC->SWTRIGR |= 0x00000000;

- Channel1 için 12 bit sağa ya da sola veya 8 bit sağa kaydedebiliriz.

## DAC channel1 12-bit right-aligned data holding register (DAC\_DHR12R1)

Address offset: 0x08

Reset value: 0x0000 0000

## DAC channel1 12-bit left aligned data holding register (DAC\_DHR12L1)

Address offset: 0x0C

Reset value: 0x0000 0000

## DAC channel1 8-bit right aligned data holding register (DAC\_DHR8R1)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DACC1DHR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

- 12 bit sağa kaydedecekiz.

Bits 11:0 DACC1DHR[11:0]: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

## RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser- ved	OTGH S	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHM A CEN	Reserved		DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved	
	rw	rw	rw	rw	rw	rw			rw	rw			rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CRCE N	Reserved			GPIOE N	GPIOH EN	GPIOG EN	GPIOF N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN	
		rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- DAC1 A4 pini olduğundan portunu sadece A portunu aktif ediyoruz.

RCC->AHB1ENR |= 0x00000001;

- RCC ve DAC1 için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

10 void RCC_Config(void)
11 {
12     RCC->CR |= 0x00010000; //HSEON
13     while(!(RCC->CR & 0x00020000)); //HSERDY
14     RCC->CR |= 0x00080000; //CSSON
15     RCC->CFGR = 0x00000000;
16     RCC->PLLCFGR |= 0x00400000; //PLLSRC
17     RCC->PLLCFGR |= 0x00000004; //PLLM 4
18     RCC->PLLCFGR |= 0x00002A00; //PLLN 168
19     RCC->PLLCFGR |= 0x00000000; //PLLP 2
20     RCC->CR |= 0x01000000; //PLLON
21     while(!(RCC->CR & 0x02000000)); //PLLRDY
22     RCC->CFGR |= 0x00000001; //SW
23     while(!(RCC->CR & 0x00000001)); //SWS
24 }

26 void DAC1_Config(void)
27 {
28     RCC->APB1ENR |= 0x20000000; //Dacen
29     RCC->AHB1ENR |= 0x00000001; //A clock enable
30
31     DAC->CR |= 0x00000001; //DAC channel1 enabled
32     DAC->SWTRIGR |= 0x00000000; //Software trigger disabled
33     DAC->DHR12R1 |= 0x00000000; //DAC channel1 12-bit right-aligned data
34 }

```

Kod Kısımlı

```
3 int i=0;
4
5 void delay(uint32_t time)
6 {
7     while(time--);
8 }
36 int main(void)
37 {
38     RCC_Config();
39     SystemCoreClockUpdate();
40     DAC1_Config();
41
42     while (1)
43     {
44         for(;i<4096; i++)
45         {
46             DAC->DHR12R1 |= i;
47             delay(16800);
48         }
49         i=0;
50     }
51 }
```

- Her i değeri arttığında i değeri kadar kaydetmiş oluyor.
- A4 pinine led bağladığımızda i değeri arttıkça voltaj değeri artacağından parlaklık artar.
- 4095 olduğunda yani 0x0000FFF olduğunda tam parlaklıktır yanar.

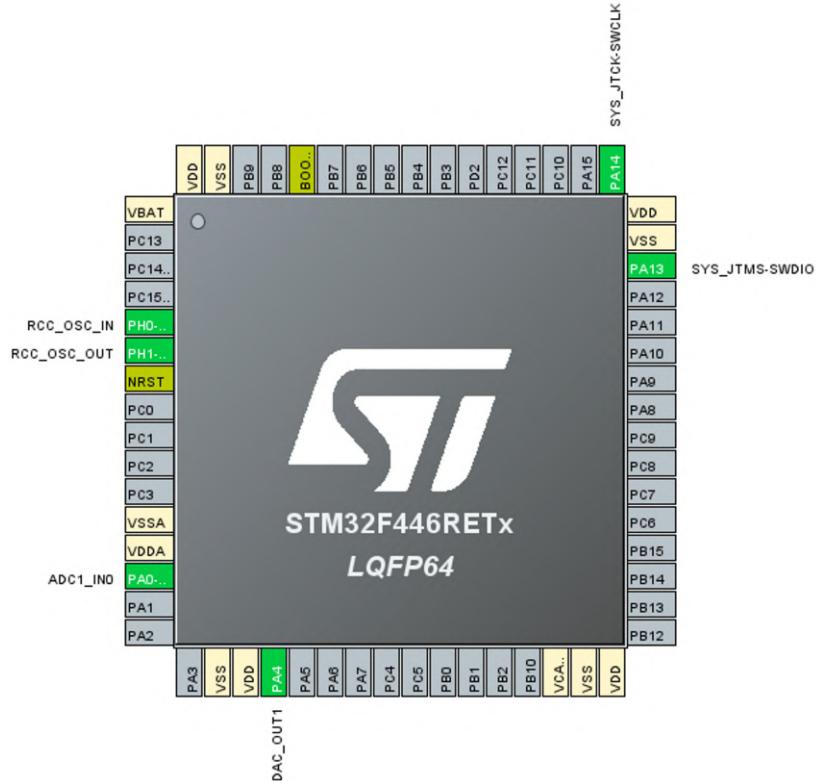
## ADC Değeri ile DAC Kontrolü

25 Aralık 2021 Cumartesi 00:57

## ADC Değeri İle DAC Kontrolü

➤ HAL

## Konfigürasyon Kısmı



Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum Value	User Label	Modified
PA0-WK...	ADC1_IN0	n/a	Analog mode	No pull-up	n/a		<input type="checkbox"/>
Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO P...	Maximum...	User L...	Modified
PA4	DAC_OUT1	n/a	Analog mode	No pull-...	n/a		<input type="checkbox"/>

- ADC1 için IN0, DAC için OUT1 seçilir.

### ADC Settings

Clock Prescaler	PCLK2 divided by 2
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

## DAC Out1 Settings

Output Buffer  
Trigger

Kod Kısmı

```
23/* Private includes -----  
24 /* USER CODE BEGIN Includes */  
25 uint16_t adc_value, dac_value;  
26 /* USER CODE END Includes */
```

- 16 bitlik okuma yapan fonksiyon yazdık.
- Geri dönüşlü fonksiyon olduğundan Return ile değeri geri döndürüyor.

```

60/* Private user code -----
61 /* USER CODE BEGIN 0 */
62uint16_t Read_ADC()
63{
64    HAL_ADC_Start(&hadc1);
65
66    if(HAL_ADC_PollForConversion(&hadc1, 100000) == HAL_OK)
67    {
68        adc_value = HAL_ADC_GetValue(&hadc1);
69    }
70    HAL_ADC_Stop(&hadc1);
71
72    return adc_value;
73}
75/* USER CODE END 0 */

107 /* USER CODE BEGIN 2 */
108 HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
109 /* USER CODE END 2 */

111 /* Infinite loop */
112 /* USER CODE BEGIN WHILE */
113 while (1)
114 {
115     /* USER CODE END WHILE */
116
117     /* USER CODE BEGIN 3 */
118     dac_value=Read_ADC();
119     HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dac_value);
120 }
121 /* USER CODE END 3 */
122 }
```

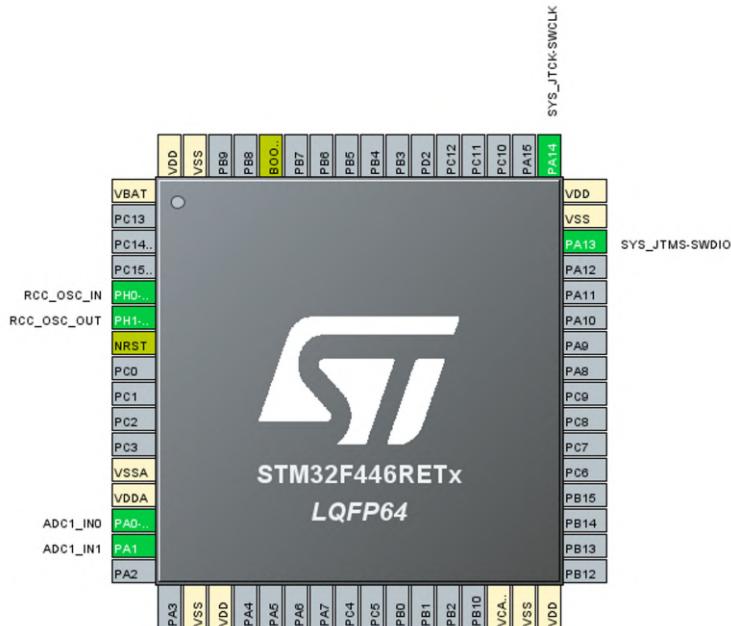
## DMA ile ADC Değer Okuma

25 Aralık 2021 Cumartesi 00:58

## DMA ile ADC Değer Okuma

➤ HAL

## Konfigürasyon Kısımları



Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum output	User Label	Modified
PA0-WKUP	ADC1_IN0	n/a	Analog mode	No pull-up	n/a		<input type="checkbox"/>
PA1	ADC1_IN1	n/a	Analog mode	No pull-up	n/a		<input type="checkbox"/>

## ADC Settings

Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

- DMA ayarları için Add diyoruz. Ardından Stream, Direction ve Priority için seçenekleri seçiyoruz. Daha sonra Direction kısmından seçtiğimiz Peripheral ve Memory için adres değişikliği için Increment Adress için işaretliyoruz. Ardından gönderilecek data boyutu belirliyoruz. 12bit çalıştığımızdan 16 bitlik olan Half Word seçimi yapıyoruz.

DMA Request	Stream	Direction	Priority
ADC1	DMA2 Stream 0	Peripheral To Memory	Very High

[Add](#)

DMA Request Settings		Peripheral	Memory
Mode	Circular	Increment Address	<input type="checkbox"/>
Use Fifo	<input type="checkbox"/>	Threshold	<input type="text"/>
		Data Width	Half Word
		Burst Size	<input type="text"/>

## Kod Kısmı

```
107 *** DMA mode IO operation ***
108 =====
109 [...]
110 (+) Start the ADC peripheral using HAL_ADC_Start_DMA(). at this stage the user specify the length
```

```

107     *** DMA mode IO operation ***
108     =====
109     [...]
110     (+) Start the ADC peripheral using HAL_ADC_Start_DMA(), at this stage the user specify the length
111         of data to be transferred at each end of conversion
112     (+) At The end of data transfer by HAL_ADC_ConvCpltCallback() function is executed and user can
113         add his own code by customization of function pointer HAL_ADC_ConvCpltCallback
114     (+) In case of transfer Error, HAL_ADC_ErrorCallback() function is executed and user can
115         add his own code by customization of function pointer HAL_ADC_ErrorCallback
116     (+) Stop the ADC peripheral using HAL_ADC_Stop_DMA()
• pData kismına veriyi tutacağımız değişkeni yazıyoruz. Bizim değişken 16 bit olarak tanımladığımızdan dönüştürmemiz
gerekıyor.
• Length ile kaç tane veri okuyacağımızı yazıyoruz.
1354④ /**
1355     * @brief Enables ADC DMA request after last transfer (Single-ADC mode) and enables ADC
1356     * peripheral
1357     * @param hadc pointer to a ADC_HandleTypeDef structure that contains
1358     * the configuration information for the specified ADC.
1359     * @param pData The destination Buffer address.
1360     * @param Length The length of data to be transferred from ADC peripheral to memory.
1361     * @retval HAL status
1362 */
1363 HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc, uint32_t* pData, uint32_t Length)

46 /* USER CODE BEGIN PV */
47 uint16_t adc_value[2];
48 /* USER CODE END PV */

94 /* USER CODE BEGIN 2 */
95 HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_value, 2);
96 /* USER CODE END 2 */

```

## ➤ REGISTER

### Konfigürasyon Kısmı

- RCC için eklemeler yaptık.

### RCC clock configuration register (RCC\_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
MCO2		MCO2 PRE[2:0]				MCO1 PRE[2:0]				I2SSCR	MCO1		RTCPRE[4:0]			
rw		rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
PPRE2[2:0]				PPRE1[2:0]				Reserved	HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	r	r	rw	rw

- AHB, APB1 ve APB2 için clock hızını böldük.

Bits 7:4 HPRE: AHB prescaler

Set and cleared by software to control AHB clock division factor.

**Caution:** The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after HPRE write.

**Caution:** The AHB clock frequency must be at least 25 MHz when the Ethernet is used.

- 0xxx: system clock not divided
- 1000: system clock divided by 2
- 1001: system clock divided by 4
- 1010: system clock divided by 8
- 1011: system clock divided by 16
- 1100: system clock divided by 64
- 1101: system clock divided by 128
- 1110: system clock divided by 256
- 1111: system clock divided by 512

```
RCC->CFG_R |= 0x00000000;
```

Bits 12:10 **PPRE1**: APB Low speed prescaler (APB1)

Set and cleared by software to control APB low-speed clock division factor.

**Caution:** The software has to set these bits correctly not to exceed 42 MHz on this domain.

The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPRE1 write.

0xx: AHB clock not divided

100: AHB clock divided by 2

101: AHB clock divided by 4

110: AHB clock divided by 8

111: AHB clock divided by 16

```
RCC->CFG_R |= 0x00001400;
```

Bits 15:13 **PPRE2**: APB high-speed prescaler (APB2)

Set and cleared by software to control APB high-speed clock division factor.

**Caution:** The software has to set these bits correctly not to exceed 84 MHz on this domain.

The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPRE2 write.

0xx: AHB clock not divided

100: AHB clock divided by 2

101: AHB clock divided by 4

110: AHB clock divided by 8

111: AHB clock divided by 16

```
RCC->CFG_R |= 0x00008000;
```

## RCC clock interrupt register (RCC\_CIR)

Address offset: 0x0C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved						CSSC	Reserved	PLL12S RDYC	PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved	PLLI2S RDYIE	PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	Reserved	PLLI2S RDYF	PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF		
	rw	rw	rw	rw	rw	rw	r		r	r	r	r	r	r		

- Kaldırılan bayrakları temizliyoruz.

Bit 19 **HSERDYC**: HSE ready interrupt clear

This bit is set by software to clear the HSERDYF flag.

0: No effect

1: HSERDYF cleared

```
RCC->CIR |= 0x00080000;
```

Bit 23 **CSSC**: Clock security system interrupt clear

This bit is set by software to clear the CSSF flag.

0: No effect

1: Clear CSSF flag

```
RCC->CIR |= 0x00800000;
```

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```

3 void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000; //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000; //CSSON
8     RCC->CFGGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000; //PLLSRC
10    RCC->PLLCFGR |= 0x00000004; //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00; //PLLN 168
12    RCC->PLLCFGR |= 0x00000000; //PLLP 2
13    RCC->CR |= 0x01000000; //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGGR |= 0x00000001; //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGGR |= 0x00000000; //HPRE AHB 1
18    RCC->CFGGR |= 0x00001400; //PPRE1 APB1 4
19    RCC->CFGGR |= 0x00008000; //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000; //HSERDYC
21    RCC->CIR |= 0x00800000; //CSSC
22 }
• GPIO için yazdığımız fonksiyon aşağıdaki gibidir.
27 void GPIO_Config(void)
28 {
29     RCC->AHB1ENR |= 0x00000001; //A clock enable
30
31     GPIOA->MODER |= 0x00000003; //PA0 Analog mode
32     GPIOA->OSPEEDR |= 0x00000003; //Very high speed
33 }

```

## ADC control register 1 (ADC\_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				OVRIE	RES		AWDEN	JAWDEN	Reserved						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCE N	DISC EN	JAUTO	AWDSG L	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Çoklu ADC okuması yapılacaksa scan yani tarama aktif edilmesi gereklidir.

Bit 8 **SCAN**: Scan mode

This bit is set and cleared by software to enable/disable the Scan mode. In Scan mode, the inputs selected through the ADC\_SQRx or ADC\_JSQRx registers are converted.

0: Scan mode disabled

1: Scan mode enabled

Note: An EOC interrupt is generated if the EOCIE bit is set:

- At the end of each regular group sequence if the EOCS bit is cleared to 0
- At the end of each regular channel conversion if the EOCS bit is set to 1

Note: A JEOC interrupt is generated only on the end of conversion of the last channel if the JEOCIE bit is set.

ADC1->CR1 |= 1 << 8;

- Çözünürlük için 12-bit kullanıyoruz.

Bits 25:24 **RES[1:0]**: Resolution

These bits are written by software to select the resolution of the conversion.

00: 12-bit (15 ADCCLK cycles)

01: 10-bit (13 ADCCLK cycles)

10: 8-bit (11 ADCCLK cycles)

11: 6-bit (9 ADCCLK cycles)

ADC1->CR1 |= 0 << 24;

## ADC control register 2 (ADC\_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
reserved	SWST ART	EXTEN			EXTSEL[3:0]				reserved	JSWST ART	JEXTEN			JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
reserved				ALIGN	EOCS	DDS	DMA	Reserved							CONT	ADON	
				rw	rw	rw	rw								rw	rw	

- ADC aktif etmek için ilk bit 1 yapıldı.

### Bit 0 ADON: A/D Converter ON / OFF

This bit is set and cleared by software.

Note: 0: Disable ADC conversion and go to power down mode  
1: Enable ADC

- Sürekli okuma yapmasını istiyoruz.

### Bit 1 CONT: Continuous conversion

This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.

0: Single conversion mode  
1: Continuous conversion mode

- DMA mod ile çalışacağız.

### Bit 8 DMA: Direct memory access mode (for single ADC mode)

This bit is set and cleared by software. Refer to the DMA controller chapter for more details.

0: DMA mode disabled  
1: DMA mode enabled

- DMA'nın sürekli çalışması için 1 yapıyorum.

### Bit 9 DDS: DMA disable selection (for single ADC mode)

This bit is set and cleared by software.

0: No new DMA request is issued after the last transfer (as configured in the DMA controller)  
1: DMA requests are issued as long as data are converted and DMA=1

- 10.bit 1 yapılır.

### Bit 10 EOCS: End of conversion selection

This bit is set and cleared by software.

0: The EOC bit is set at the end of each sequence of regular conversions. Overrun detection is enabled only if DMA=1.  
1: The EOC bit is set at the end of each regular conversion. Overrun detection is enabled.

- Çevrim başlaması için 30.biti 1 yapıyorum.

### Bit 30 SWSTART: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as the conversion starts.

0: Reset state  
1: Starts conversion of regular channels

Note: This bit can be set only when ADON = 1 otherwise no conversion is launched.

ADC1->CR2 |= 1 << 0;

ADC1->CR2 |= 1 << 1;

ADC1->CR2 |= 1 << 8;

ADC1->CR2 |= 1 << 9;

ADC1->CR2 |= 1 << 10;

ADC1->CR2 |= 1 << 30;

- Kaç farklı kanaldan çevrim yapacağımızı belirtmemiz gerekiyor. Biz sadece A portun 0.bitinden yani 1 kanal'dan okuma yapacağım.

## ADC regular sequence register 1 (ADC\_SQR1)

Address offset: 0x2C

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										L[3:0]			SQ16[4:1]			
										rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

## ADC regular sequence register 1 (ADC\_SQR1)

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										L[3:0]	SQ16[4:1]				
										rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]						SQ14[4:0]						SQ13[4:0]		
rw	rw	rw	rw	rw	rw	rw	rw						rw	rw	rw

## ADC regular sequence register 2 (ADC\_SQR2)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ12[4:0]					SQ11[4:0]					SQ10[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10_0	SQ9[4:0]						SQ8[4:0]						SQ7[4:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

## ADC regular sequence register 3 (ADC\_SQR3)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0	SQ3[4:0]						SQ2[4:0]						SQ1[4:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- SQR1 için 20.biti 0 yaparak 1 tane kanal olduğunu belirtiyoruz.

Bits 23:20 L[3:0]: Regular channel sequence length

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.  
0000: 1 conversion  
0001: 2 conversions

...  
1111: 16 conversions

ADC1->SQR1 |= 0 << 20;

- SQR3 için SQ1 kısmına kanal numarasını yazıyoruz.

Bits 4:0 SQ1[4:0]: 1st conversion in regular sequence

ADC1->SQR3 |= 0 << 0;

## ADC common control register (ADC\_CCR)

Address offset: 0x04 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TSVREFE	VBATE	Reserved			
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA[1:0]	DDS	Res.	DELAY[3:0]					Reserved				MULTI[4:0]			
rw	rw	rw	rw	rw	rw	rw	rw					rw	rw	rw	rw

### Bits 17:16 ADCPRE: ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADC. The clock is common for all the ADCs.

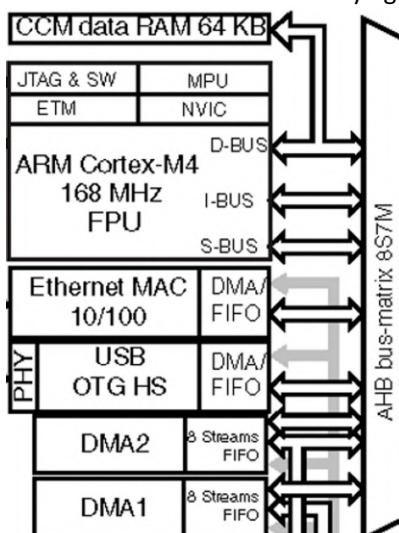
- Note: 00: PCLK2 divided by 2
- 01: PCLK2 divided by 4
- 10: PCLK2 divided by 6
- 11: PCLK2 divided by 8

ADC->CCR |= 0x00010000;

- ADC için yazdığımız fonksiyon aşağıdaki gibidir.

```
32 void ADC_Config(void)
33 {
34     RCC->APB2ENR |= 0x00000100; //ADC1
35
36     ADC1->CR1 |= 1 << 8; //SCAN
37     ADC1->CR1 |= 0 << 24; //RES 12-bit
38     ADC1->CR2 |= 1 << 0; //ADON
39     ADC1->CR2 |= 1 << 1; //CONT
40     ADC1->CR2 |= 1 << 8; //DMA
41     ADC1->CR2 |= 1 << 9; //DDS
42     ADC1->CR2 |= 1 << 10; //EOCS
43     ADC1->CR2 |= 1 << 30; //SWSTART
44     ADC1->SMPR2 |= 0x00000003; //SMP0 56 cycles
45     ADC1->SQR1 |= 0 << 20; //1 conversion
46     ADC1->SQR3 |= 0 << 0; //SQ1
47     ADC->CCR |= 0x00010000; //ADCPRE 4
48 }
```

- DMA'nın clock hattı AHB'ye gidiyor.



### RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser- ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHM A CEN	Reserved		DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved		
	rw	rw	rw	rw	rw	rw			rw	rw			rw			
Reserved	CRCE N	Reserved				GPIOE N	GPIOH EN	GPIOG EN	GPIOF N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN	Reserved	
						rw	rw	rw	rw	rw	rw	rw	rw			

## Bit 22 DMA2EN: DMA2 clock enable

Set and cleared by software.

0: DMA2 clock disabled

1: DMA2 clock enabled

- 22.pini 1 yapıyoruz.

RCC->[AHB1ENR](#) |= 0x00400000;

- ADC için DMA2 kullanıyoruz. Tabloya baktığımızda biz burada Stream 4'ü seçiyoruz.

**DMA1 request mapping**

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_RX
Channel 1	I2C1_RX	-	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX <sup>(1)</sup>	UART7_TX <sup>(1)</sup>	TIM3_CH4 TIM3_UP	UART7_RX <sup>(1)</sup>	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX <sup>(1)</sup>	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

- These requests are available on STM32F42xxx and STM32F43xxx only.

**DMA2 request mapping**

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A <sup>(1)</sup>	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A <sup>(1)</sup>	ADC1	SAI1_B <sup>(1)</sup>	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B <sup>(1)</sup>	SPI6_TX <sup>(1)</sup>	SPI6_RX <sup>(1)</sup>	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX <sup>(1)</sup>	SPI5_TX <sup>(1)</sup>	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	-	SPI1_TX	-	-
Channel 4	SPI4_RX <sup>(1)</sup>	SPI4_TX <sup>(1)</sup>	USART1_RX	SDIO	-	USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX <sup>(1)</sup>	SPI4_TX <sup>(1)</sup>	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX <sup>(1)</sup>	SPI5_TX <sup>(1)</sup>	TIM8_CH4 TIM8_TRIG TIM8_COM

- These requests are available on STM32F42xxx and STM32F43xxx.

## DMA stream x configuration register (DMA\_SxCR) (x = 0..7)

This register is used to configure the concerned stream.

Address offset: 0x10 + 0x18 × stream number

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			CHSEL[2:0]			MBURST [1:0]		PBURST[1:0]		Reser- ved	CT	DBM	PL[1:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PINCOS	MSIZE[1:0]	PSIZE[1:0]	MINC	PINC	CIRC	DIR[1:0]	PFCTRL	TCIE	HTIE	TEIE	DMEIE	EN			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

- 0.bit 0 olduğunda konfigürasyon yapmaya imkan tanıyor. Bunun için okuma yapacağız.

**Bit 0 EN:** Stream enable / flag stream ready when read low

This bit is set and cleared by software.

0: Stream disabled

1: Stream enabled

This bit may be cleared by hardware:

- on a DMA end of transfer (stream ready to be configured)
- if a transfer error occurs on the AHB master buses
- when the FIFO threshold on memory AHB port is not compatible with the size of the burst

When this bit is read as 0, the software is allowed to program the Configuration and FIFO bits registers. It is forbidden to write these registers when the EN bit is read as 1.

*Note: Before setting EN bit to '1' to start a new transfer, the event flags corresponding to the stream in DMA\_LISR or DMA\_HISR register must be cleared.*

- 1 olduğu sürece 0 olana kadar bekliyoruz. 0 olunca döngüden çıkış alt satırı geçecek.

```
while((DMA2_Stream4->CR & 0x00000001) == 1);
```

- ADC'den RAM'e yazacağımız bunun için Peripheral-to-memory seçiyoruz.

**Bits 7:6 DIR[1:0]:** Data transfer direction

These bits are set and cleared by software.

00: Peripheral-to-memory

01: Memory-to-peripheral

10: Memory-to-memory

11: reserved

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 0 << 6;
```

- Sürekli transfer için 8.bit'i 1 yapıyoruz.

**Bit 8 CIRC:** Circular mode

This bit is set and cleared by software and can be cleared by hardware.

0: Circular mode disabled

1: Circular mode enabled

When the peripheral is the flow controller (bit PFCTRL=1) and the stream is enabled (bit EN=1), then this bit is automatically forced by hardware to 0.

It is automatically forced by hardware to 1 if the DBM bit is set, as soon as the stream is enabled (bit EN ='1').

```
DMA2_Stream4->CR |= 1 << 8;
```

- Adresin değişmesini istemiyoruz. 9.bit'i 0 yapıyoruz.

**Bit 9 PINC:** Peripheral increment mode

This bit is set and cleared by software.

0: Peripheral address pointer is fixed

1: Peripheral address pointer is incremented after each data transfer (increment is done according to PSIZE)

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 0 << 9;
```

**Bit 10 MINC:** Memory increment mode

This bit is set and cleared by software.

0: Memory address pointer is fixed

1: Memory address pointer is incremented after each data transfer (increment is done according to MSIZE)

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 1 << 10;
```

- Peripheral ve Memory için data boyutunu 32 bit belirliyoruz.

**Bits 12:11 PSIZE[1:0]:** Peripheral data size

These bits are set and cleared by software.

00: Byte (8-bit)

01: Half-word (16-bit)

10: Word (32-bit)

11: reserved

These bits are protected and can be written only if EN is '0'

#### Bits 14:13 **MSIZE[1:0]**: Memory data size

These bits are set and cleared by software.

- 00: byte (8-bit)
- 01: half-word (16-bit)
- 10: word (32-bit)
- 11: reserved

These bits are protected and can be written only if EN is '0'.

In direct mode, MSIZE is forced by hardware to the same value as PSIZE as soon as bit EN = '1'.

```
DMA2_Stream4->CR |= 2 << 11; //PSIZE Word (32-bit)  
DMA2_Stream4->CR |= 2 << 13; //MSIZE Word (32-bit)
```

- Önceliği çok yüksek yapıyoruz.

#### Bits 17:16 **PL[1:0]**: Priority level

These bits are set and cleared by software.

- 00: Low
- 01: Medium
- 10: High
- 11: Very high

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 3 << 16;
```

- Kanal seçimi için 0.kanal yapıyoruz.

#### Bits 27:25 **CHSEL[2:0]**: Channel selection

These bits are set and cleared by software.

- 000: channel 0 selected
- 001: channel 1 selected
- 010: channel 2 selected
- 011: channel 3 selected
- 100: channel 4 selected
- 101: channel 5 selected
- 110: channel 6 selected
- 111: channel 7 selected

These bits are protected and can be written only if EN is '0'

- Kaç tane çevresel birimden yani ADC'den okuma yaptığımızı belirtiyoruz.

### DMA stream x number of data register (DMA\_SxNDTR) (x = 0..7)

Address offset:  $0x14 + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Bits 15:0 **NDT[15:0]**: Number of data items to transfer

Number of data items to be transferred (0 up to 65535). This register can be written only when the stream is disabled. When the stream is enabled, this register is read-only, indicating the remaining data items to be transmitted. This register decrements after each DMA transfer.

Once the transfer has completed, this register can either stay at zero (when the stream is in normal mode) or be reloaded automatically with the previously programmed value in the following cases:

- when the stream is configured in Circular mode.
- when the stream is enabled again by setting EN bit to '1'

If the value of this register is zero, no transaction can be served even if the stream is enabled.

- Biz bir tane birimden yani ADC1'den okuma yapıyoruz.

```
DMA2_Stream4->NDTR |= 1;
```

- Okuma yaptığımız Çevresel birimin adresini yazıyoruz.

## DMA stream x peripheral address register (DMA\_SxPAR) (x = 0..7)

Address offset:  $0x18 + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PAR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **PAR[31:0]**: Peripheral address

Base address of the peripheral data register from/to which the data will be read/written.

These bits are write-protected and can be written only when bit EN = '0' in the DMA\_SxCR register.

DMA transfer.

Once the transfer has completed, this register can either stay at zero (when the stream is in normal mode) or be reloaded automatically with the previously programmed value in the following cases:

- when the stream is configured in Circular mode.
- when the stream is enabled again by setting EN bit to '1'

If the value of this register is zero, no transaction can be served even if the stream is enabled.

## ADC regular data register (ADC\_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in [Figure 48](#) and [Figure 49](#).

```
DMA2_Stream4->PAR |= (uint8_t) &ADC1->DR;
```

- Değişkenin adresini yazıyoruz.

## DMA stream x memory 0 address register (DMA\_SxM0AR) (x = 0..7)

Address offset:  $0x1C + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MOA[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MOA[31:0]**: Memory 0 address

Base address of Memory area 0 from/to which the data will be read/written.

These bits are write-protected. They can be written only if:

- the stream is disabled (bit EN= '0' in the DMA\_SxCR register) or
- the stream is enabled (EN='1' in DMA\_SxCR register) and bit CT = '1' in the DMA\_SxCR register (in Double buffer mode).

```

3 uint8_t adc;
4 uint8_t adc1[8];

DMA2_Stream4->M0AR |= (uint8_t) &adc1;

```

### DMA stream x FIFO control register (DMA\_SxFCR) (x = 0..7)

Address offset:  $0x24 + 0x24 \times \text{stream number}$

Reset value: 0x0000 0021

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								FEIE	Reser ved	FS[2:0]			DMDIS	FTH[1:0]	
								r	r	r	r	r	r	r	r

Bits 1:0 FTH[1:0]: FIFO threshold selection

These bits are set and cleared by software.

- 00: 1/4 full FIFO
- 01: 1/2 full FIFO
- 10: 3/4 full FIFO
- 11: full FIFO

These bits are not used in the direct mode when the DMIS value is zero.

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->FCR |= 0 << 1;
```

- Stream4'ü fonksiyonun en son satırında aktif ediyoruz.

```
DMA2_Stream4->CR |= 1 << 0;
```

- Adres tanımları fonksiyonun ortalarında yazmak yerine while döngüsünün sonrasında yazdık.
- DMA için yazdığımız fonksiyon aşağıdaki gibidir.

```

53 void DMA_Config(void)
54 {
55     RCC->AHB1ENR |= 0x00400000; //DMA2
56
57     while((DMA2_Stream4->CR & 0x00000001) == 1);
58     DMA2_Stream4->PAR |= (uint8_t) &ADC1->DR; //PAR Peripheral address
59     DMA2_Stream4->M0AR |= (uint8_t) &adc1; //M0A Memory 0 address
60     DMA2_Stream4->CR |= 0 << 6; //DIR
61     DMA2_Stream4->CR |= 1 << 8; //CIRC
62     DMA2_Stream4->CR |= 0 << 9; //PINC
63     DMA2_Stream4->CR |= 1 << 10; //PSIZE Word (32-bit)
64     DMA2_Stream4->CR |= 2 << 11; //MSIZE Word (32-bit)
65     DMA2_Stream4->CR |= 2 << 13; //PL
66     DMA2_Stream4->CR |= 3 << 16; //CHSEL
67     DMA2_Stream4->CR |= 0 << 25; //NDT
68     DMA2_Stream4->NDTR |= 1; //FTH 1/2 full FIFO
69     DMA2_Stream4->FCR |= 0 << 1; //EN Stream enabled
70     DMA2_Stream4->CR |= 1 << 0;
71 }

```

#### Kod Kısmı

- ADC için çevrimi DMA'dan sonra başlatmak için 80.satırı tekrar yazdık.

```

73 int main(void)
74 {
75     RCC_Config();
76     GPIO_Config();
77     ADC_Config();
78     DMA_Config();
79     //ADC1->CR2 |= ADC_CR2_SWSTART;
80     ADC1->CR2 |= 0x40000000;
81
82     while (1)
83     {
84
85     }
86 }

```

# Timer Değer Okuma

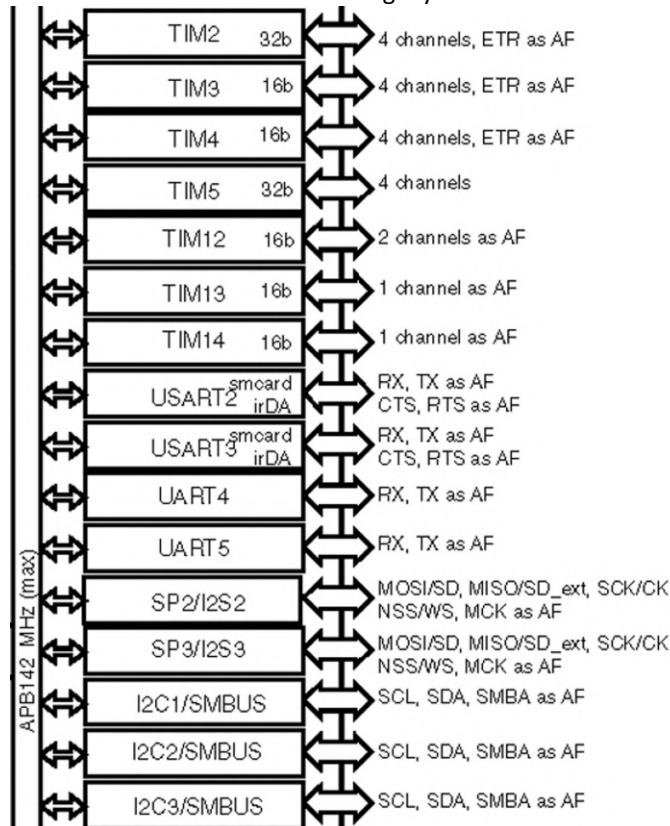
25 Aralık 2021 Cumartesi 00:59

## Timer Değer Okuma

### ➤ REGISTER

#### Konfigürasyon Kismı

- TIM2'nin clock hattı APB1'e gitiyor.



### RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		

0.biti 1 yapıyoruz.

Bit 0 **TIM2EN**: TIM2 clock enable

Set and cleared by software.

0: TIM2 clock disabled

1: TIM2 clock enabled

RCC->APB1ENR |= 0x00000001;

## TIMx control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN		
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		

- Sayma yapacağımızdan saymayı başlatmak için 0.bit'i aktif ediyoruz. Sayma başlayacağından bu işlem fonksiyonun en son satırında olmalı.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled  
1: Counter enabled

*Note:* External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

`TIM2->CR1 |= 1 << 0;`

- Saymanın yönünü belirliyoruz. Biz yukarı doğru saymasını istiyoruz.

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter  
1: Counter used as downcounter

*Note:* This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

`TIM2->CR1 |= 0 << 4;`

- 5. ve 6.biti 0 yapıyoruz.

Bits 6:5 **CMS**: Center-aligned mode selection

- 00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).  
01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting down.  
10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting up.  
11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set both when the counter is counting up or down.

*Note:* It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

- Sistem clock 168MHz ayarlamıştık ve TIM2 clock veri yolu ise 42MHz'dir fakat bunun 2 katı değeri alıyorlardı yani 84MHz'dir. Biz bunu bölmek istiyorsak ayarlayabiliyoruz. Biz bölmüyoruz.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00:  $t_{DTS} = t_{CK\_INT}$   
01:  $t_{DTS} = 2 \times t_{CK\_INT}$   
10:  $t_{DTS} = 4 \times t_{CK\_INT}$   
11: Reserved

- 8. ve 9.biti 0 yapıyoruz.

`TIM2->CR1 |= 0 << 8;`

## TIMx slave mode control register (TIMx\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Slave mode çalışmayaçğız.

Bits 2:0 **SMS**: Slave mode selection

Bits 2:0 **SMS**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1 then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI2FP1 edge depending on TI1FP2 level.

010: Encoder mode 2 - Counter counts up/down on TI1FP2 edge depending on TI2FP1 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS=100).*

*Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

*The clock of the slave timer must be enabled prior to receiving events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

```
TIM2->SMCR |= 0 << 0;
```

## TIMx event generation register (TIMx\_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TG	Res.	CC4G	CC3G	CC2G	CC1G	UG	
								w		w	w	w	w	w	

- Sayma dolduğunda sıfırlaması için 1 yaparız. 0.bit 1 yapıyoruz.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx\_ARR) if DIR=1 (downcounting).

```
TIM2->EGR |= 1 << 0;
```

## TIMx prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Prescaler bizim sayısının en üst seviyesini belirler.

- Sayma işlemi 0'dan başlamayıp 1'den başladığından 1 eksigini alarak yazarız.

- TIM2 clock hızı 84MHz olduğundan bunu kaça bölmeliyim diye soruyoruz. Bu clock hız için 42000'e bölüyoruz. Bu sayı da 41999 sayısı yapıyor.

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in "reset mode").

```
TIM2->PSC |= 41999;
```

- 84MHz'i 42000'e böldüğümüzde 2000 sayısı yapıyor. Bu değer auto-reload oluyor. Bunun 1 eksigini yazıyoruz.

## TIMx auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2 and TIM5).

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 18.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

```
TIM2->ARR |= 1999;
```

- Böylece 1 sn'de 2000'e kadar sayıyor.
- RCC ve TIM için yazdığımız fonksiyonlar aşağıdaki gibidir.

```
5 void RCC_Config(void)
6 {
7     RCC->CR |= 0x00010000; //HSEON
8     while(!(RCC->CR & 0x00020000)); //HSERDY
9     RCC->CR |= 0x00080000; //CSSON
10    RCC->CFGGR = 0x00000000;
11    RCC->PLLCFGR |= 0x00400000; //PLLSRC
12    RCC->PLLCFGR |= 0x00000004; //PLLM 4
13    RCC->PLLCFGR |= 0x00002A00; //PLLN 168
14    RCC->PLLCFGR |= 0x00000000; //PLLP 2
15    RCC->CR |= 0x01000000; //PLLON
16    while(!(RCC->CR & 0x02000000)); //PLLRDY
17    RCC->CFGGR |= 0x00000001; //SW
18    while(!(RCC->CR & 0x00000001)); //SWS
19    RCC->CFGGR |= 0x00000000; //HPRE AHB 1
20    RCC->CFGGR |= 0x00001400; //PPRE1 APB1 4
21    RCC->CFGGR |= 0x00008000; //PPRE2 APB2 2
22    RCC->CIR |= 0x00080000; //HSERDYC
23    RCC->CIR |= 0x00800000; //CSSC
24 }

26 void TIM_Config(void)
27 {
28     RCC->APB1ENR |= 0x00000001; //TIM2EN
29
30     TIM2->CR1 |= 0 << 4; //DIR Counter used as up counter
31     TIM2->CR1 |= 0 << 5; //CMS Edge-aligned mode
32     TIM2->CR1 |= 0 << 8; //tDTS = tCK_INT 84MHz
33     TIM2->SMCR |= 0 << 0; //SMS Slave mode disabled
34     TIM2->EGR |= 1 << 0; //UG Update generation
35     TIM2->PSC |= 41999; //PSC Prescaler value
36     TIM2->ARR |= 1999; //ARR Auto-reload value
37     TIM2->CR1 |= 1 << 0; //CEN Counter enabled
38 }
```

Kod Kısmı

## TIMx counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- CNT registeri sayma işlemini burada tutuyor.
- Değişken atayıp bu değişkene CNT registerine eşitliyoruz.

```
3 uint16_t count = 0;
```

```
40 int main(void)
41 {
42     RCC_Config();
43     TIM_Config();
44
45     while (1)
46     {
47         count = TIM2->CNT;
48     }
50 }
```

Variable Name	Address/Expression	Read Value
count	0x2000002c	1838

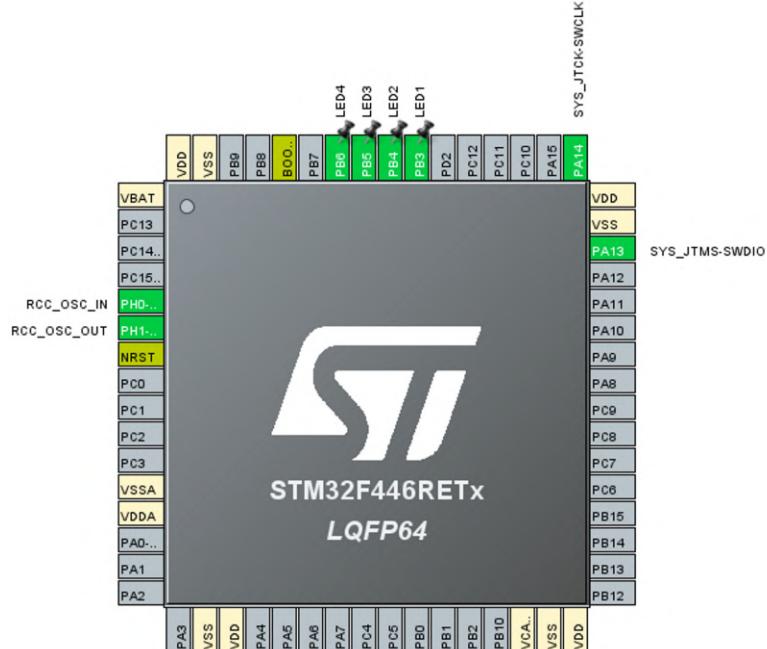
# Timer Interrupt

25 Aralık 2021 Cumartesi 00:59

# Timer Interrupt

➤ HAL

## Konfigürasyon Kısımları

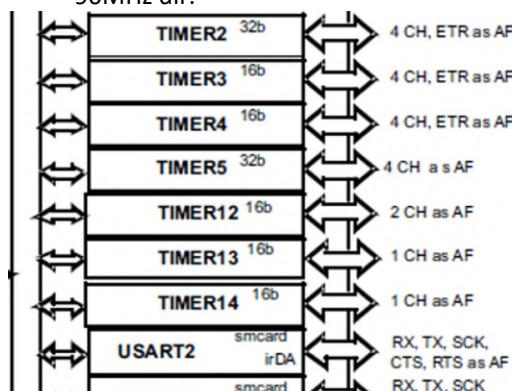


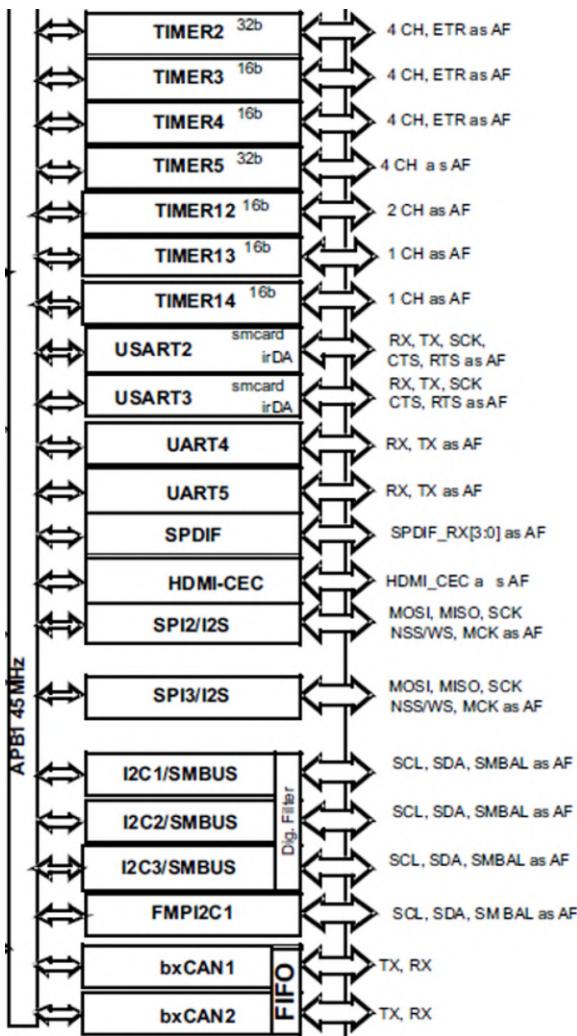
Pin N...	Signal on ...	GPIO out...	GPIO mode	GPIO Pull...	Maximum ...	User Label	Modified
PB3	n/a	Low	Output Pu...	No pull-up...	Very High	LED1	<input checked="" type="checkbox"/>
PB4	n/a	Low	Output Pu...	No pull-up...	Very High	LED2	<input checked="" type="checkbox"/>
PB5	n/a	Low	Output Pu...	No pull-up...	Very High	LED3	<input checked="" type="checkbox"/>
PB6	n/a	Low	Output Pu...	No pull-up...	Very High	LED4	<input checked="" type="checkbox"/>

- Timers kısmından TIM2 seçimi yapılır. Ardından Mod kısmından sadece Clock Source kısmını Internal Clock yaparız.

Slave Mode	Disable	▼
Trigger Source	Disable	▼
Clock Source	Internal Clock	▼
Channel1	Disable	▼
Channel2	Disable	▼
Channel3	Disable	▼
Channel4	Disable	▼
Combined Channels	Disable	▼

- Sistem clock 180MHz ayarlamıştık ve TIM2 clock veri yolu ise 45MHz'dir. fakat bunun 2 katı değeri alıyorlardı yani 90MHz'dır.





- Daha sonra Parameter Settings'den TIM2 1 saniye aralıklarla tekrarlı şekilde çalıştıracak değerler girilir. Sayma işlemi 0'dan başlamayıp 1'den başladığından 1 eksigini alarak yazarız.
  - Prescalar bizim sayısının en üst seviyesini belirler. Burası 16 bit olduğundan en fazla 65535 yazabilirim. TIM2 clock hızı 90MHz olduğundan bunu kaça bölmeliyim diye soruyoruz. Bu clock hız için 45000'e böölüyoruz. Bu sayı da 44999 sayısını yapıyor.
  - 90MHz'i 45000'e böldüğümüzde 2000 sayısı yapıyor. Bu değer Auto-reload oluyor. Bunun da 1 eksigini yazıyoruz. Counter Period kısmında her seferinde taşıma işlemi bittikten sonra tekrar bunu yükler. Yükleyeceği değeri yazarız.
  - Sayma şeklini Up belirleyip yukarı doğru sayıyor.
  - Auto-reload preload kısmında Enabled diyerek sayma bittiğinde başa dönmesini sağlarız.
  - 2000 değeri yazmasaydık sonuç 2000 Hz olacağından sonucunda 0,0005s yani 1 sn=1000 ms olmak üzere 0,5 ms olacaktır.

## Counter Settings

Prescaler (PSC - 16 bits value)	45000-1
Counter Mode	Up
Counter Period (AutoReload Register -	2000-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Enable

$$UpdateEvent = \frac{Timer_{clock}}{(Prescaler + 1)(Period + 1)}$$

$$UpdateEvent = \frac{90.000.000}{(45000)(2000)} = 1 \text{ Hz} = \frac{1}{1} s = 1 \text{ s}$$

- NVIC Settings kısmından TIM2 global interrupt Enabled yapılır. Bununla her güncellemede, sayıyı bitirmede bir interrupt olmasını sağlıyoruz.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	<input checked="" type="checkbox"/>	0	0

Kod Kısmı

- hal\_tim.c dosyasından Timer'ı Interrupt ile başlatmamız gerekiyor.

```

453 /**
454  * @brief Starts the TIM Base generation in interrupt mode.
455  * @param htim TIM Base handle
456  * @retval HAL status
457 */
458 HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)

```

```

95  /* USER CODE BEGIN 2 */
96  HAL_TIM_Base_Start_IT(&htim2);
97  /* USER CODE END 2 */

```

- 96.satır çalışınca it.c dosyasına gider. Burada TIM için geçerli fonksiyonu 1sn'de bir çalıştırır.

```

202 /**
203  * @brief This function handles TIM2 global interrupt.
204 */
205 void TIM2_IRQHandler(void)
206 {
207  /* USER CODE BEGIN TIM2_IRQHandler_0 */
208
209  /* USER CODE END TIM2_IRQHandler_0 */
210  HAL_TIM_IRQHandler(&htim2);
211  /* USER CODE BEGIN TIM2_IRQHandler_1 */
212
213  /* USER CODE END TIM2_IRQHandler_1 */
214 }

```

- 210.satır ile bu fonksiyon dallanır ve dallanan fonksiyon içerisinde Callback fonksiyonu vardır. Bu fonksiyon sayesinde int main içerisinde de kodlarımı yazabilirim.

```

2038 /**
2039  * @defgroup TIM_Exported_Functions_Group9 TIM Callbacks functions
2040  * @{
2041  */
2042 /* Callback in non blocking modes (Interrupt and DMA) *****/
2043 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
    • 1sn aralıklarla ledi yakıp söndürüyor.
57 /**
58  * Private user code -----
59  * USER CODE BEGIN 0 */
60 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
61 {
62     HAL_GPIO_TogglePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
63 }
64 /* USER CODE END 0 */

```

# Timer ile Delay Oluşturma

Friday, July 7, 2023 10:56 AM

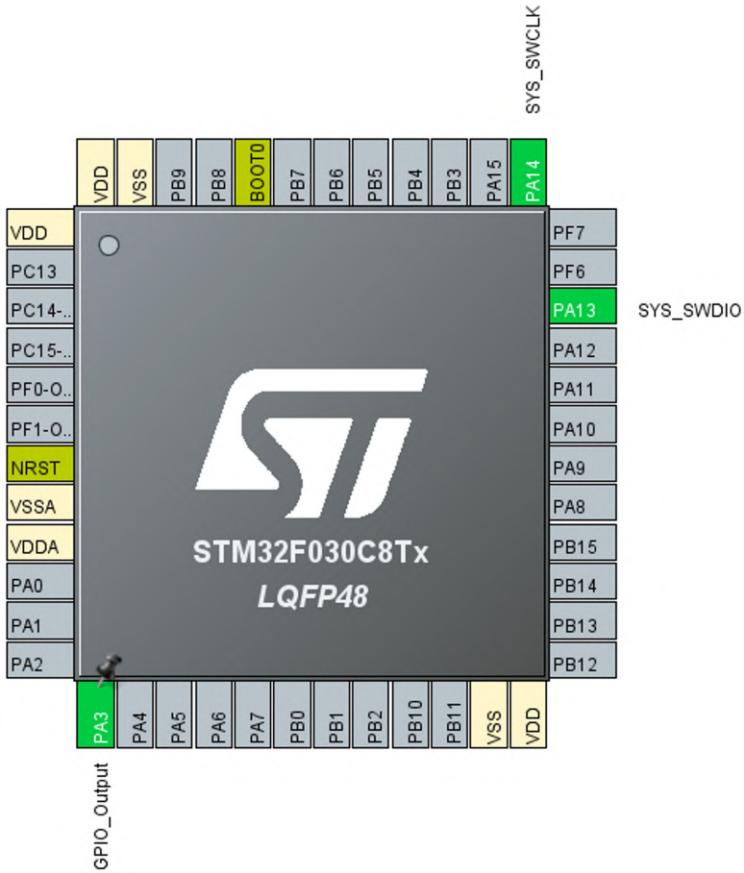
## Timer ile Delay Oluşturma

### ➤ HAL

#### Teori

- <https://controllerstech.com/create-1-microsecond-delay-stm32/> ve <https://deepbluemedded.com/stm32-delay-microsecond-millisecond-utility-dwt-delay-timer-delay/> linkleri kaynak olarak kullanabiliriz.
- HAL\_Delay fonksiyonuyla minimum 1 milisaniye(ms) gecikme oluşturabiliyoruz fakat 1 mikrosaniye(us) gecikme oluşturan bir fonksiyon bulunmuyor. Bunu yapmak için Timer birimin zamanlayıcısını kullanacağız.
- Öncelikle bir Timer birimi seçmemiz gerekiyor. Bu işlem için özel bir Timer seçmemize gerek yok. Seçim sonrası bağlı olduğu clock hattını bilmemiz gerekiyor.

#### Konfigürasyon Kısımlı



- Timer1 birimini kullanacağımız ve bu birim APB1 clock hattına bağlıdır.
- Mode kısmında Clock Source için Internal Clock seçimi yapıyoruz.
- Prescaler değerine APB1 clock hattının değerini yazıyoruz. Counter Period değerine yazılabilen maksimum değer olan 16 bitin karşılığı 65535 değerini yazıyoruz.

#### Counter Settings

Prescaler (PSC - 16 bits value)	48-1
Counter Mode	Up
Counter Period (AutoReload R...	65535
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bi...	0
auto-reload preload	Disable

#### Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

#### Kod Kısımlı

- Milisaniye ve mikrosaniye için fonksiyon yazıyoruz.
- delay\_us() fonksiyonu için önce TIM1'deki sayaçın ilk değeri 0 ayarlanır. Sonra parametre olarak girdiğimiz 16 bitlik us değişkenine ulaşana kadar saymaya devam eder.

```

218 /* USER CODE BEGIN 4 */
219 @void delay_us(uint16_t us)
220 {
221     __HAL_TIM_SET_COUNTER(&htim1,0);
222     while(__HAL_TIM_GET_COUNTER(&htim1)<us);
223 }
224
225 @void delay_ms(uint16_t ms)
226 {
227     while(ms > 0)
228     {
229         __HAL_TIM_SET_COUNTER(&htim1,0);
230         ms--;
231         while(__HAL_TIM_GET_COUNTER(&htim1) < 1000);
232     }
233 }
234 /* USER CODE END 4 */
• Ana döngüye girmeden önce timer başlatılır.
91    /* USER CODE BEGIN 2 */
92    HAL_TIM_Base_Start(&htim1);
93    /* USER CODE END 2 */
• Delay için delay_us() ya da delay_ms() fonksiyonlarını istediğimiz gibi kullanabiliriz.
97    while (1)
98    {
99        /* USER CODE END WHILE */
100
101       /* USER CODE BEGIN 3 */
102       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
103       delay_ms(1000);
104       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
105       delay_ms(1000);
106   }
107   /* USER CODE END 3 */

```

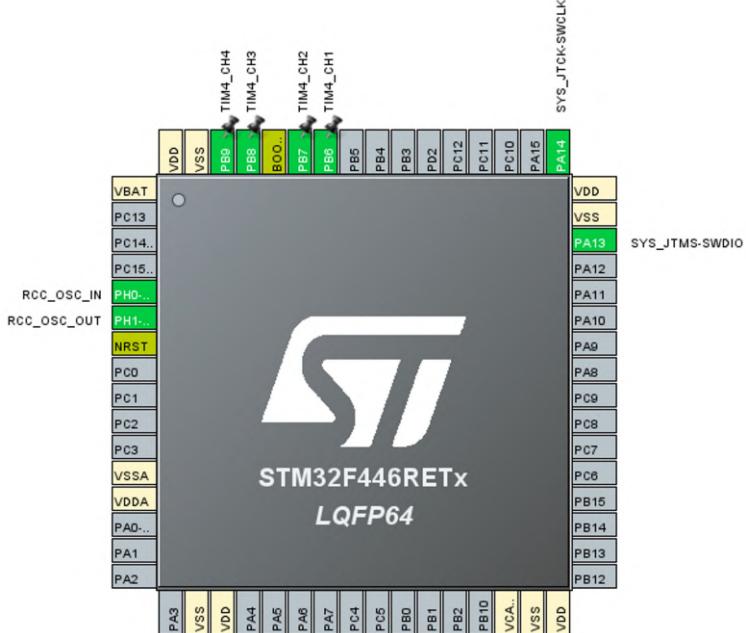
## PWM Kullanımı

25 Aralık 2021 Cumartesi 00:59

## PWM Kullanımı

## ➤ HAL

## Konfigürasyon Kısımları



Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum output	User Label	Modified
PB6	TIM4_CH1	n/a	Alternate Function	No pull-up or down	Low		<input type="checkbox"/>
PB7	TIM4_CH2	n/a	Alternate Function	No pull-up or down	Low		<input type="checkbox"/>
PB8	TIM4_CH3	n/a	Alternate Function	No pull-up or down	Low		<input type="checkbox"/>
PB9	TIM4_CH4	n/a	Alternate Function	No pull-up or down	Low		<input type="checkbox"/>

- Timers kısmından TIM4 seçimi yapılır. Ardından Mod kısmından kanal seçimi yapılır.

Slave Mode	Disable	▼
Trigger Source	Disable	▼
<input type="checkbox"/> Internal Clock		
Channel1	PWM Generation CH1	▼
Channel2	PWM Generation CH2	▼
Channel3	PWM Generation CH3	▼
Channel4	PWM Generation CH4	▼
Combined Channels	Disable	▼

- Period değerimize göre kanal çıkışlarına Pulse değeri yazacağız.
  - Period kısmını Duty Cycle en fazla 100 olduğundan Period kısmına 100-1 olarak gireriz. Yani Period 100 ve Pulse değeri 50 girersek aslında %50 Duty Cycle olur.
  - 10kHz için işlem sonucunda Prescaler 90000 girilir.
  - $1 \text{ kHz} = 1000 \text{ Hz}$

$$UpdateEvent = \frac{90.000.000}{(Prescaler + 1)(100)} = 10000\ Hz = 10\ kHz$$

$$Prescaler + 1 = 90$$

#### Counter Settings

Prescaler (PSC - 16 bits value)	90-1
Counter Mode	Up
Counter Period (AutoReload Register - 16 b.	100-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

#### PWM Generation Channel 1

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

#### PWM Generation Channel 2

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

#### PWM Generation Channel 3

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

#### PWM Generation Channel 4

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

- Mode 1 ile Mode 2 arasındaki fark High ile Low durumlarının tersi olmasıydı.
- Mode 1 durumunda kanal %25 ise %25'te High %75'te Low oluyordu.  
%75 olduğunda %25'te Low %75'te High oluyor.
- Fast Mode durumun Disable olduğu zaman saymaya yaparken üst limite kadar sayıdıktan sonra 0'a doğru azala azala inerken Enabled olduğu zaman 0'a doğru birden iner.

#### Kod Kısmı

- hal\_tim.c dosyasından Timer'ı PWM ile başlatmamız gerekiyor.

```
1439 /**
1440 * @brief Starts the PWM signal generation.
1441 * @param htim TIM handle
1442 * @param Channel TIM Channels to be enabled
1443 *          This parameter can be one of the following values:
1444 *          @arg TIM_CHANNEL_1: TIM Channel 1 selected
1445 *          @arg TIM_CHANNEL_2: TIM Channel 2 selected
1446 *          @arg TIM_CHANNEL_3: TIM Channel 3 selected
1447 *          @arg TIM_CHANNEL_4: TIM Channel 4 selected
1448 * @retval HAL status
1449 */
1450 HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)

91 /* USER CODE BEGIN 2 */
92 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
93 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
94 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
95 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
96 /* USER CODE END 2 */

45 /* USER CODE BEGIN PV */
46 int i;
47 /* USER CODE END PV */
```

```

1375 /**
1376  * @brief Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.
1377  * @param __HANDLE__ TIM handle.
1378  * @param __CHANNEL__ TIM Channels to be configured.
1379  *   This parameter can be one of the following values:
1380  *     @arg TIM_CHANNEL_1: TIM Channel 1 selected
1381  *     @arg TIM_CHANNEL_2: TIM Channel 2 selected
1382  *     @arg TIM_CHANNEL_3: TIM Channel 3 selected
1383  *     @arg TIM_CHANNEL_4: TIM Channel 4 selected
1384  * @param __COMPARE__ specifies the Capture Compare register new value.
1385  * @retval None
1386 */
1387 #define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__) \
1388 (((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1 = (__COMPARE__)) :\\
1389 ((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2 = (__COMPARE__)) :\\
1390 ((__CHANNEL__) == TIM_CHANNEL_3) ? ((__HANDLE__)->Instance->CCR3 = (__COMPARE__)) :\\
1391 ((__HANDLE__)->Instance->CCR4 = (__COMPARE__)))
• Compare kısmına Pulse değerini yazıyoruz.
98  /* Infinite loop */
99  /* USER CODE BEGIN WHILE */
100 while (1)
101 {
102  /* USER CODE END WHILE */
103
104  /* USER CODE BEGIN 3 */
105  for(i=0;i<=1999;i++)
106  {
107      __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,i);
108      __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_2,i);
109      __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,i);
110      __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_4,i);
111      HAL_Delay(100);
112  }
113
114 }
115 /* USER CODE END 3 */
116 }

```

## ➤ REGISTER

### Konfigürasyon Kısımları

- RCC için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

3 void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;          //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;   //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;   //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;   //PLL2
13    RCC->CR |= 0x01000000;        //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;      //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000;      //HPRE AHB 1
18    RCC->CFGR |= 0x00001400;      //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000;      //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000;       //HSERDYC
21    RCC->CIR |= 0x00800000;       //CSSC
22 }

```

- GPIO için çıkışlarını alternatif fonksiyon yapıyoruz.

## GPIO port mode register (GPIOx\_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

## GPIO port mode register (GPIOx\_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Kart üzerindeki ledleri kullanıyoruz. 24., 26., 28. ve 30. biti 1 yapıyoruz.

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

GPIOD->**MODER** |= 2 << 24 | 2 << 26 | 2 << 28 | 2 << 30;

- Alternatif fonksiyon ile kast edilen pinin çevresel birimlerden I2C, SPI olarak kullanılacağını belirtiyor. Biz burada TIM4 için kullanacağımızı belirteceğiz.

## GPIO alternate function low register (GPIOx\_AFRL) (x = A..I/J/K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

## GPIO alternate function high register (GPIOx\_AFRH) (x = A..I/J)

(x = A..I/J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Low ile 0 ile 7.pinler arası iken high 8 ile 15.pinler arasıdır. Biz 12, 13, 14 ve 15. pinleri kullandığımızdan high olanı kullanıyoruz.

Bits 31:0 **AFRHy**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

AFRHy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14

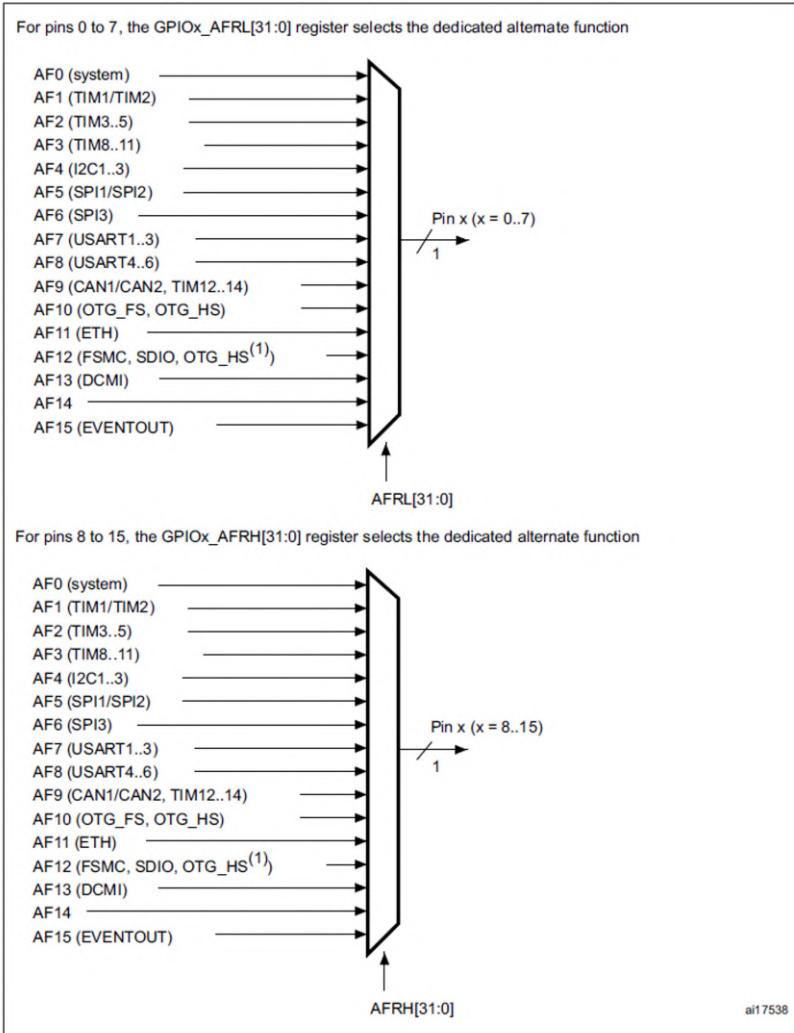
Bits 31:0 **AFRH<sub>y</sub>**: Alternate function selection for port x bit y (y = 8..15)  
 These bits are written by software to configure alternate function I/Os

AFRH<sub>y</sub> selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

- Seçtiğimiz TIM4 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitapçığına bakıyoruz.

#### Selecting an alternate function



- TIM4, AF2'de bulunuyor. Böylece pinlere AF2 için olan 0010 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunda belirtmemiz gerekiyor. AFR'ye Ctrl ile sağ tıklarız. AFR'nin parantez içinde 2 elemanlı dizi olduğunu gösterir. 0.eleman low, 1.eleman high temsil eder. Biz 1 yazıyoruz.

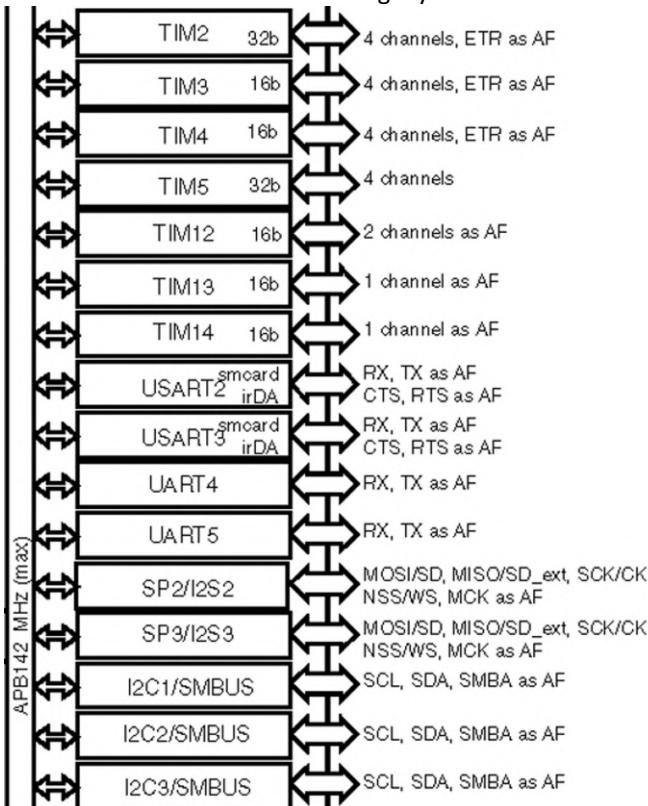
```
682     __IO uint32_t AFR[2]; /*!< GPIO alternate function registers, Address offset: 0x20-0x24 */
```

```
GPIOD->AFR[1] |= 2 << 16 | 2 << 20 | 2 << 24 | 2 << 28;
```

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```
24 void GPIO_Config(void)
25 {
26     RCC->AHB1ENR |= 0x00000008; //D clock enable
27
28     GPIOD->MODER |= 2 << 24 | 2 << 26 | 2 << 28 | 2 << 30; //PD12, PD13, PD14, PD15
29     GPIOD->AFR[1] |= 2 << 16 | 2 << 20 | 2 << 24 | 2 << 28; //TIM4 AFRH12, AFRH13, AFRH14, AFRH15
30     GPIOD->OTYPER |= 0x00000000; //Output push-pull
31     GPIOD->OSPEEDR |= 0xFF000000; //Very high speed
32     GPIOD->PUPDR |= 0x00000000; //No pull-up, pull-down
33 }
```

- TIM4'nin clock hattı APB1'e gidiyor.



## RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

- 2.biti 1 yapıyoruz.

Bit 2 **TIM4EN**: TIM4 clock enable

Set and cleared by software.

0: TIM4 clock disabled

1: TIM4 clock enabled

RCC->**APB1ENR** |= 0x00000004; //TIM4EN

## TIMx control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved						CKD[1:0]		ARPE	CMS			DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Sayma yapacağımdan saymayı başlatmak için 0.biti aktif ediyoruz. Sayma başlayacağından bu işlem fonksiyonun en son satırında olmalı.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled
- 1: Counter enabled

*Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.*

CEN is cleared automatically in one-pulse mode, when an update event occurs.

**TIM4->CR1 |= 1 << 0;**

- Saymanın yönünü belirliyoruz. Biz yukarı doğru saymasını istiyoruz.

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter
- 1: Counter used as downcounter

*Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.*

**TIM4->CR1 |= 0 << 4;**

- 5. ve 6.biti 0 yapıyoruz.

Bits 6:5 **CMS**: Center-aligned mode selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set both when the counter is counting up or down.

*Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)*

**TIM4->CR1 |= 0 << 5;**

- Sistem clock 168MHz ayarlamıştık ve TIM4 clock veri yolu ise 42MHz'dir fakat bunun 2 katı değeri alıyorlardı yani 84MHz'dir. Biz bunu bölmek istiyorsak ayarlayabiliyoruz. Biz bölmüyoruz.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

- 8. ve 9.biti 0 yapıyoruz.

**TIM4->CR1 |= 0 << 8;**

- PWM aslında capture/compare mode kısmına giriyor.

## **TIMx capture/compare mode register 1 (TIMx\_CCMR1)**

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. Take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]		OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]		OC1PE	OC1FE	CC1S[1:0]			
IC2F[3:0]		IC2PSC[1:0]					IC1F[3:0]		IC1PSC[1:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- 1.kanal için output yapıyoruz yani ilk 2 bit 0 yapıyoruz.

#### Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

- Mod olarak PWM mode 1 seçiyoruz. Bunun için bite 110 yazıyoruz.

#### Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF=1).

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.*

- 2.kanal için output yapıyoruz. 8. ve 9.bit 0 yapıyoruz.

#### Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).*

- 2.kanalın modunu 1.kanal'da yaptığımız gibi PWM Mode 1 yapıyoruz.

#### Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

```
TIM4->CCMR1 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12;
```

## TIMx capture/compare mode register 2 (TIMx\_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]			OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]
	IC4F[3:0]				IC4PSC[1:0]					IC3F[3:0]				IC3PSC[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- CCMR2 ile bu sefer kanal 3 ve 4 için yapıyoruz. Kanal 1 ve 2 ile yaptıklarımızın aynısını yapıyoruz.

#### Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = 0 in TIMx\_CCER).*

#### Bits 6:4 **OC3M**: Output compare 3 mode

#### Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = 0 in TIMx\_CCER).*

#### Bits 14:12 **OC4M**: Output compare 4 mode

```
TIM4->CCMR2 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12;
```

## TIMx capture/compare enable register (TIMx\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw

- Çıkışları Enabled yapıyoruz.

0., 4., 8. ve 12. bitleri 1 yapıyoruz.

#### Bit 0 **CC1E**: Capture/Compare 1 output enable.

##### CC1 channel configured as output:

0: Off - OC1 is not active

1: On - OC1 signal is output on the corresponding output pin

##### CC1 channel configured as input:

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled

1: Capture enabled

#### Bit 4 **CC2E**: Capture/Compare 2 output enable.

refer to CC1E description

#### Bit 8 **CC3E**: Capture/Compare 3 output enable.

refer to CC1E description

#### Bit 12 **CC4E**: Capture/Compare 4 output enable.

refer to CC1E description

```
TIM4->CCER |= 1 << 0 | 1 << 4 | 1 << 8 | 1 << 12;
```

## TIMx prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Prescaler bizim sayısının en üst seviyesini belirler.

- Sayma işlemi 0'dan başlamayıp 1'den başladığından 1 eksigini alarak yazarız.

- TIM4 clock hızı 84MHz olduğundan bunu kaça bölmeliyim diye soruyoruz. Bu clock hız için 42000'e bölüyoruz. Bu sayı da 41999 sayısı yapıyor.

#### Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

TIM4->PSC |= 41999;

- 84MHz’i 42000'e böldüğümüzde 2000 savısı yapıyor. Bu değer auto-reload oluyor. Bunun 1 eksigini yazıyoruz.

## **TIMx auto-reload register (TIMx\_ARR)**

Address offset: 0x2C

Reset value: 0xFFFF FFFF

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2 and TIM5).

Bits 15:0    **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 18.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

TIM4->ARR |= 1999;

- Böylece 1 sn'de 2000'e kadar sayıyor.
  - Pinlere atayacağımız paslar CCR1 12.pin, CCR2 13.pin, CCR3 14.pin ve CCR4 15.pin içindir.

## TIMx capture/compare register 1 (TIMx\_CCR1)

Address offset: 0x34

Reset value: 0x0000 0000

## TIMx capture/compare register 2 (TIMx CCR2)

Address offset: 0x38

Reset value: 0x0000 0000

## TIMx capture/compare register 3 (TIMx\_CCR3)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR3[31:16] (depending on timers)															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro

## TIMx capture/compare register 4 (TIMx\_CCR4)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR4[31:16] (depending on timers)															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro

- Bunlara atayacağımız değerler en fazla Auto-reload değeri kadar olabilir.

TIM4->CCR1 |= 500;

TIM4->CCR2 |= 1000;

TIM4->CCR3 |= 1500;

TIM4->CCR4 |= 1999;

- TIM için yazdığımız fonksiyonlar aşağıdaki gibidir.

```
35 void TIM_Config(void)
36 {
37     RCC->APB1ENR |= 0x00000004;                                //TIM4EN
38
39     TIM4->CR1 |= 0 << 4;                                     //DIR Counter used as up counter
40     TIM4->CR1 |= 0 << 5;                                     //CMS Edge-aligned mode
41     TIM4->CR1 |= 0 << 8;                                     //tDTS = tCK_INT 84MHz
42     TIM4->CCMR1 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12;   //Capture/Compare selected output, PWM mode 1 (1 & 2)
43     TIM4->CCMR2 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12;   //output, PWM mode 1 selected (3 & 4)
44     TIM4->CCER |= 1 << 0 | 1 << 4 | 1 << 8 | 1 << 12;    //output enable (1, 2, 3 & 4 )
45     TIM4->PSC |= 41999;                                      //PSC Prescaler value
46     TIM4->ARR |= 1999;                                       //ARR Auto-reload value
47     TIM4->CCR1 |= 500;                                       //PD12
48     TIM4->CCR2 |= 1000;                                      //PD13
49     TIM4->CCR3 |= 1500;                                      //PD14
50     TIM4->CCR4 |= 1999;                                      //PD15
51     TIM4->CR1 |= 1 << 0;                                     //CEN Counter enabled
52 }
```

### Kod Kısmı

```
54 int main(void)
55 {
56     RCC_Config();
57     GPIO_Config();
58     TIM_Config();
59
60     while (1)
61     {
62
63 }
64 }
```

- Pindeki ledlerin durumu 12.pin %25, 13.pin %50, 14.pin %75, 15.pin %100 parlaklıktta yanıyor.

# UART ile Mesaj Gönderme

16 Haziran 2023 Cuma 10:44

## UART ile Mesaj Gönderme

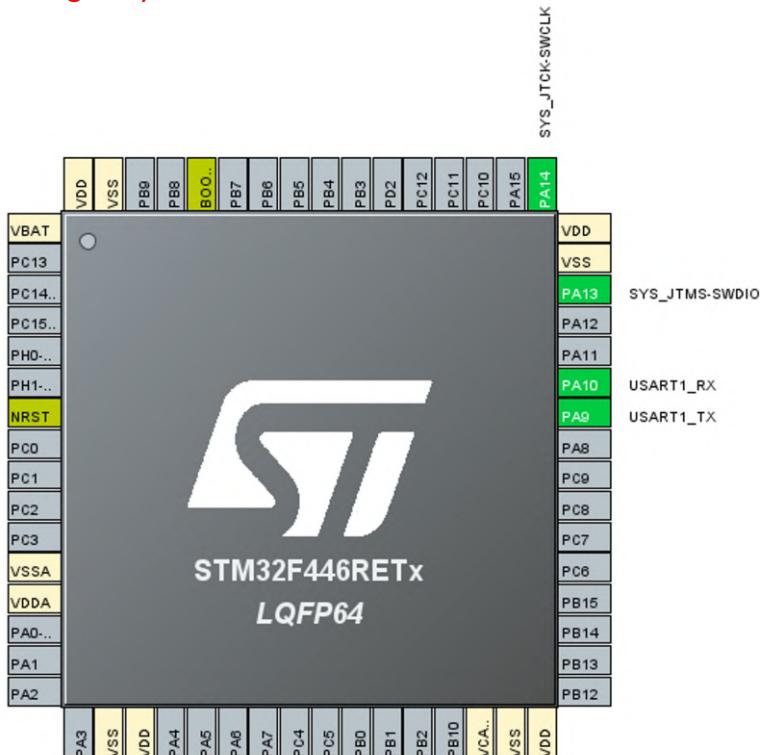
### ➤ HAL

#### Teori

- TTL, dijital mantık devrelerinde yaygın olarak kullanılan bir elektriksel sinyal standartıdır. Düşük gerilim seviye olarak 0V ile yüksek gerilim seviyeleri için 3.3V veya 5V kullanır.
- İletişimi USB, RS232 veya RS485 üzerinden sağlamak için TTL dönüştürücü kullanmamız gereklidir. USB için FT232, RS232 için MAX3232 veya MAX232, RS485 için MAX485 gibi entegreleri kullanabiliriz.
- Nucleo kartlarının USART2 pini ile bağlantı yapıldığında, terminal ile haberleşmeyi kartın üzerindeki USB ile sağlayabiliyoruz. USART2 dışındaki diğer USART pinlerini kullanırsak bu pinlerin çıkışına TTL dönüştürücü kullanmamız gereklidir.
- Seri Port işlemlerinde kullandığınız TTL dönüştürücü bilgisayarınızda gözükmüyorsa çipin driverini yüklemeniz gerekiyor. [FT232RL](#) için linkten driver indirip aygit yöneticisinden güncelleyebilirsiniz.
- Printf komutu ile veri göndermek için <https://www.youtube.com/watch?v=SpTh30wTmcM> ile <https://www.youtube.com/watch?v=WnCpPf7u4Xo> linklerindeki videoları inceleyebiliriz.
- Printf fonksiyonu ve türevleri hakkında bilgi almak için <https://bilgisayarkavramlari.com/2012/05/31/printf-sprintf-fprintf/> link üzerinden bakabiliriz.

#### Polling

#### Konfigürasyon Kısımları



Pin N...	Signal on Pin	GPIO out...	GPIO mo...	GPIO Pull...	Maximum...	User L...	Modified
PA2	USART2_RX	n/a	Alternate ...	No pull-up...	Very High	<input type="checkbox"/>	
PA3	USART2_TX	n/a	Alternate ...	No pull-up...	Very High	<input type="checkbox"/>	

Mode Asynchronous ▼

- Sadece Transmit yapacağımızdan Data Direction kısmından Transmit Only seçeneğini seçebiliriz.

#### Basic Parameters

Baud Rate	9600 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

#### Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples

### Kod Kısımları

- Sprintf komutu için stdio.h, strlen kütüphanesi için string.h kütüphanesini ekledik.

```
20 /* Includes -----  
21 #include "main.h"  
22 #include "stdio.h"  
23 #include "string.h"  
24 /* Private includes
```

- Göndereceğimiz ifadeleri tutacağımız char değişkenli dizi ekliyoruz.

```
46 /* USER CODE BEGIN PV */  
47 char tx_buff[50];  
48 uint8_t vize=65,final=87;  
49 float ortalama;  
50 /* USER CODE END PV */
```

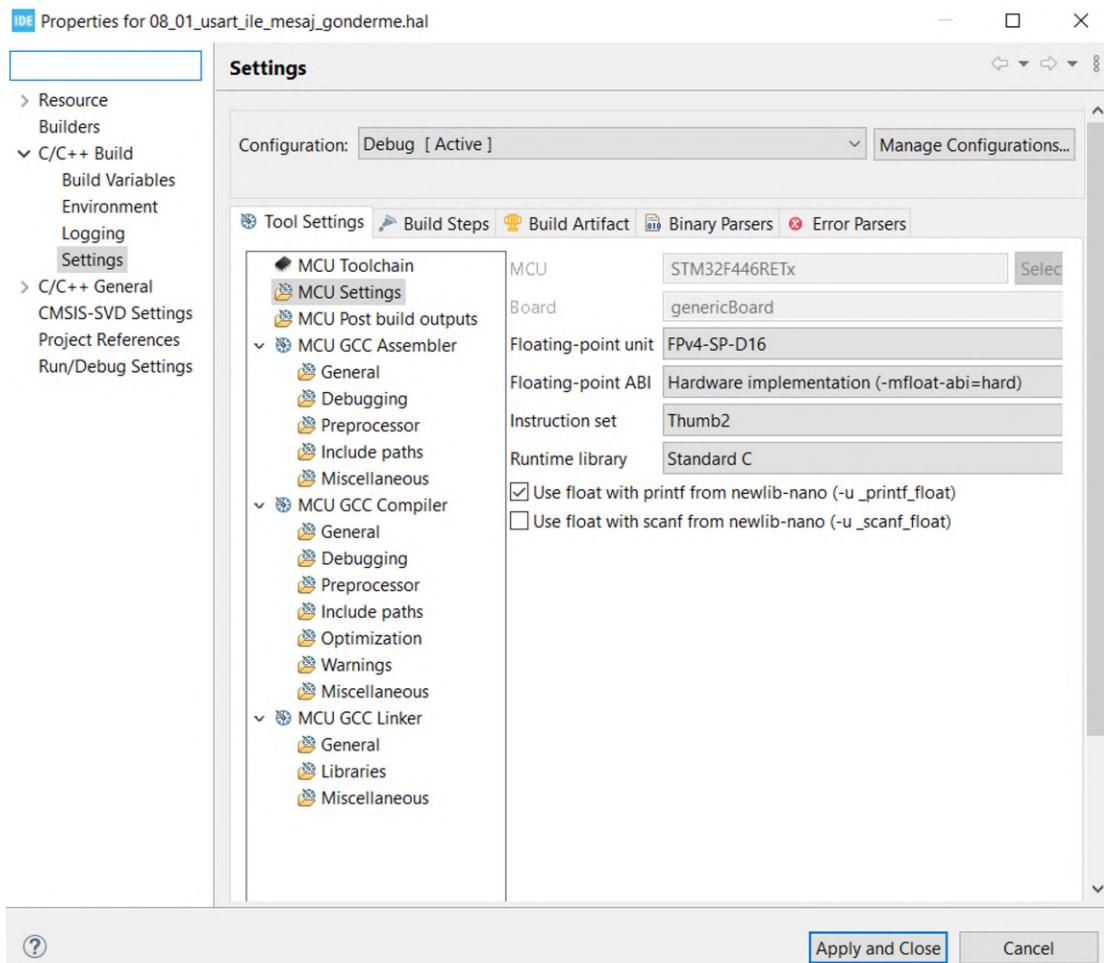
- Sprintf komutu ile yazdırma yaparken fonksiyon içinde de kullanabiliriz ya da ayrı satırda yazıp daha sonra fonksiyon içinde boyutunu belirtmek için strlen fonksiyonunu kullanırız.
- Strlen yerine sizeof fonksiyonu da kullanabilir .
- \n, metinde bir alt satıra geçmek için \r ise metinde paragraf başı yapmak için kullanılır.

```
109 /* USER CODE BEGIN 2 */  
110 ortalama=(vize * 0.4) + (final * 0.6);  
111 sprintf(tx_buff,"Vize:%d, Final:%d, Ortalama:%.2f\r\n",vize,final,ortalama);  
112 /* USER CODE END 2 */
```

- USART işleminde ilk olarak polling mode için HAL\_UART\_Transmit() fonksiyonunu kullanacağız.
- Fonksiyon için 4 parametre giriyoruz. Birincisi UART ayarlarının tutulduğu veri yapısı, ikincisi gönderilecek veri, üçüncüsü verinin kaç karakter göndereceğimiz yani kaç byte olduğu ve son olarak dördüncüsü veri aktarımının tamamlanması için beklenen maksimum süredir.
- uint8\_t tipinde bir data göndermemizi istiyor fakat biz char değişkeni kullandığımızdan bir çevrim yapmamız gerekiyor.

```
102 /* USER CODE BEGIN WHILE */  
103 while (1)  
104 {  
105 /* USER CODE END WHILE */  
106  
107 /* USER CODE BEGIN 3 */  
108 HAL_UART_Transmit(&huart2, (uint8_t *)tx_buff, strlen(tx_buff), 100);  
109 HAL_Delay(500);  
110 }  
111 /* USER CODE END 3 */  
112 }
```

- Float değeri yazdırmak için bir ayar yapılmalıdır. Bunun için proje dosyasına sağ tıklayıp Properties tıklanır. C/C++ Build kısmından Settings kısmına gelinir ve buradan Tool Settings kısmından MCU Settings kısmından -u\_printf\_float kutucuğu işaretlenir.



- Timeout süresi, milisaniye cinsinden belirtilir. Kod kısmında timeout süresi 50 olarak belirledik. Bu, veri gönderme işleminin en fazla 50 milisaniye sürmesi gerektiği anlamına gelir. Eğer belirtilen süre içinde veri gönderimi tamamlanamazsa, iletişim timeout olur ve ilgili hata durumu işlenebilir.
- Timeout süresini ihtiyaçlarınıza göre ayarlayabiliriz. Daha uzun süreler belirlemek, daha yavaş veya yoğun bir iletişim ortamında güvenilirlik sağlayabilir. Ancak, çok uzun timeout süreleri kullanmak, iletişim hatalarını algılama ve hızlı bir şekilde hata durumlarına yanıt verme yeteneğini azaltabilir.
- Timeout süresini öğrenmek için farklı yollara başvurabiliriz.  
Biz Hal\_GetTick() fonksiyonunu kullanarak ne kadarlık bir veri gönderme süresi olduğunu buluruz. Bize sonuç verirken 1ms gecikme ekler.
- Kodu aşağıdaki gibi düzenleriz. Int değişken türünde time1 ve time değişkenlerini globalde atarız. Daha sonra fonksiyonun dönüşünden gelen değeri önceki satırda gelen değeri çıkararak sonuca ulaşırız.
- Debug girdiğimizde time değişkenine baktığımızda 36 değeri döndürüyor. Çıkan sonuctan 1 çıkarırız. Sonuç olarak toplamda 35ms'de veriyi gönderiyor. Böylece Timeout için girdiğimiz 50 değeri yeterlidir.
- Hesap yaparak öğrenmek istersek 35 byte veri gönderdiğimizi biliyoruz. Bu değeri bir byte için geçen süre ile çarparsak buluruz.  
9600 baud rate için bir byte'in iletiminde geçen süre  $1/9600$  den çıkan sonucu 10 ile çarparız daha sonra gönderilen byte'teki adeti olan 35 ile çarparsak sonucunda 35ms olduğunu buluruz.
- Sonuç olarak Polling metodunu kullanarak ana döngüde transmit yaparken 35ms bekleme yapıyor. Bu bekleme döngünün çalışırken başka yapılan bir işi yavaşılatır.

```

105     while (1)
106     {
107         /* USER CODE END WHILE */
108
109         /* USER CODE BEGIN 3 */
110         timel = HAL_GetTick();
111
112         HAL_UART_Transmit(&huart1, (uint8_t *)tx_buff, strlen(tx_buff), 50);
113
114         time = HAL_GetTick() - timel;
115
116         HAL_Delay(500);
117     }
118     /* USER CODE END 3 */
119 }

```

- Printf komutu ile yazdırma istersek aşağıdaki kodu ekleriz. Bu kod ile while döngüsünde Transmit fonksiyonunu yazmamıza gerek kalmıyor.

```

60 /* USER CODE BEGIN 0 */
61 int _write(int file, char *ptr, int len)
62 {
63     HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, HAL_MAX_DELAY);
64     return len;
65 }
66 /* USER CODE END 0 */

112     while (1)
113     {
114         /* USER CODE END WHILE */
115
116         /* USER CODE BEGIN 3 */
117         //HAL_UART_Transmit (&huart1, tx_buff, strlen(tx_buff), 100);
118         printf("Vize: %d, Final:%d, Ortalama:%.2f\r\n",vize,final,ortalama);
119         HAL_Delay(500);
120     }
121     /* USER CODE END 3 */
122 }

```

### Interrupt

#### Konfigürasyon Kısmı

- Eğer Interrupt kullanacaksak aktif edilmesi gereklidir.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART1 global interrupt	<input checked="" type="checkbox"/>	0	0

#### Kod Kısmı

- USART işleminde interrupt mode olarak HAL\_UART\_Transmit\_IT() fonksiyonunu kullanacağız. Böylece döngünün ana akışını kesmeden veriyi yollamış olacağız.  
Ayrıca HAL\_UART\_TxCpltCallback, geri çağrıma fonksiyonu ile UART iletişim modülü tarafından bir veri iletim işlemi tamamlandığında otomatik olarak çağırarak yapılması gereken işlemleri gerçekleştirebiliriz.
- Interruptı kullanırken iki türlü kullanabiliriz. Birincisi, Transmit fonksiyonu ana döngü içerisinde yazıp çalıştırabiliriz daha sonra kesmeye girdiğinde Callback fonksiyonu ile başka işlemler gerçekleştirebiliriz.

```

123     HAL_UART_Transmit_IT(&huart1, tx_buff, strlen(tx_buff));
124 }
125 /* USER CODE END 3 */
126 }

```

- İkincisinde ana döngüye girmeden bir kere Transmit edip daha sonra Callback fonksiyonuyla transfer işlemi gerçekleştirebiliriz.
- Burada döngüden her çıktığında kesmeye gidiyor

```

62 /* USER CODE BEGIN 0 */
63 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
64 {
65     HAL_UART_Transmit_IT(&huart1, tx_buff, strlen(tx_buff));
66 }

```

```

110     HAL_UART_Transmit_IT(&huart1, tx_buff, strlen(tx_buff));
111     /* USER CODE END 2 */

```

- Her iki durumda da ana döngü bloke olmamış olacaktır.

## DMA

### Konfigürasyon Kısmı

- Eğer DMA kullanacaksak aktif edilmesi gereklidir.
- Dairesel modda, DMA verileri iletmeye devam edecektir. Tüm verileri ilettikten sonra, otomatik olarak baştan başlayacaktır.
- Hafızada sadece 1 byte yer kaplayan karakterleri gönderdiğimiz için Data With Byte olarak seçilmiştir.

DMA Request	Stream	Direction	Priority
USART1_TX	DMA2 Stream 7	Memory To Peripheral	Low

Add    Delete

DMA Request Settings

	Peripheral	Memory
Mode	Circular	<input checked="" type="checkbox"/>
Increment Address	<input checked="" type="checkbox"/>	
Use Fifo	<input type="checkbox"/>	
Threshold		
Data Width	Byte	Byte
Burst Size		

### Kod Kısmı

- USART işleminde DMA mode olarak HAL\_UART\_Transmit\_DMA() fonksiyonunu kullanacağız.
- DMA ayrıca kesme ile aynı şekilde çalışır. Böylece kesme için kullanabildiğimiz Callback fonksiyonlarını kullanabiliriz.

```

131     HAL_UART_Transmit_DMA(&huart1, tx_buff, strlen(tx_buff));
132 }
133 /* USER CODE END 3 */
134 }

```

- Ana döngüye girmeden hem de Callback fonksiyonu çağrımadan da gönderme işlemi yapabiliriz.

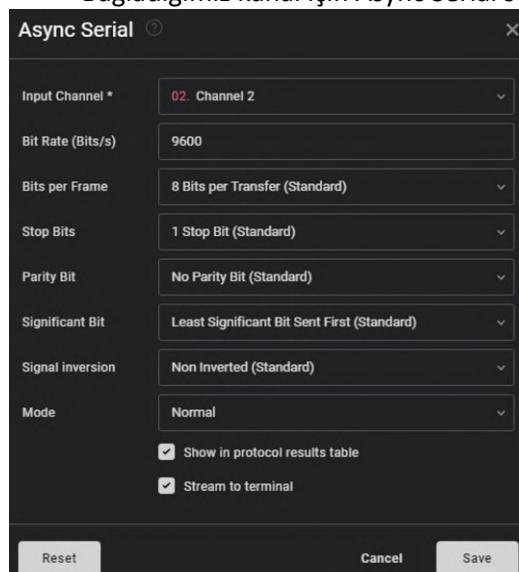
```

115     HAL_UART_Transmit_DMA(&huart1, tx_buff, strlen(tx_buff));
116 /* USER CODE END 2 */

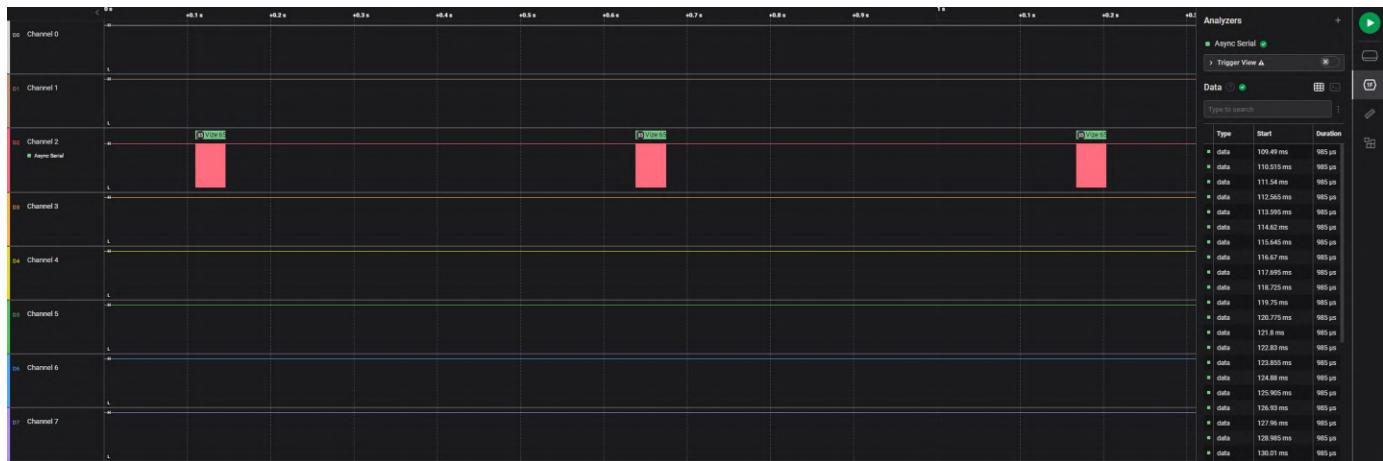
```

### Logic Analyzer Kısmı

- Ayrıca seri port dışında logic analyzer ile bakaraka sonuçları elde edebiliriz.  
Bağladığımız kanal için Async Serial seçimi yapıp benzer ayarlamaları yapıyoruz.



- İşlemcinin Tx ucunu Logic Analyzer'da Channel2'ye bağladık.
- Sonuç olarak 500ms aralıklarla 35 byte veriyi 35ms'de yolluyor.
- Ayrıca veri göndermeyi 50 ms altında yaparak işlemci zaman aşımına uğramıyor. Eğer kod kısmında 50 değil de 35'in altında bir değer yazsaydık veri kaybı olacaktı.



- Verileri görüntüülerken gelendataları ASCII olarak görüntüülüyoruz.

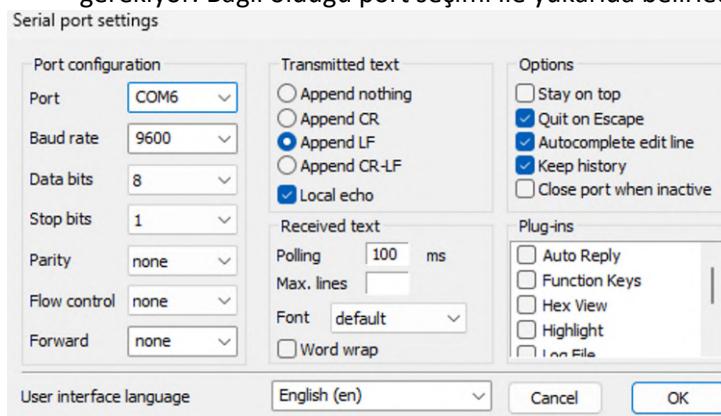


- Gelen verideki 5 verisine detaylı bakalım. 5'i ASCII'den Binary'e çevirmemiz gerekiyor.
- Ekrandan de değiştirebiliriz ya da <https://www.rapidtables.com/convert/number/ascii-to-binary.html> linkten bu işlemi yapabiliriz.
- Ayrıca [https://www.netdunyasi.com/blog/uploads/images/202105/image\\_750x\\_60985f549fe84.jpg](https://www.netdunyasi.com/blog/uploads/images/202105/image_750x_60985f549fe84.jpg) linkten ASCII tablosuna bakabiliriz.
- Binary'e çevirdiğimizde 00110101 değerini görürüz.
- İlk ve son bitler start ve stop bit olmak üzere nokta ile belirlmiş 8 bitlik data ile birlikte toplamda 10 bit var.
- Burada sırayla bakıldığından binary değeri ile aynı sonucu görmüş oluruz.

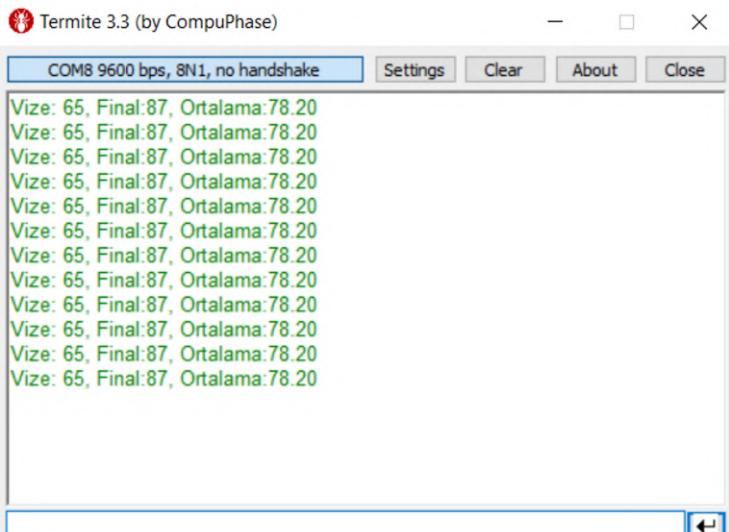


### Seri Port Kısı

- Seri port yazdırma için Termite programını kullanıyoruz. Programı açtıktan sonra bazı ayarları yapmamız gerekiyor. Bağlı olduğu port seçimi ile yukarıda belirlediğimiz 9600 baud rate değerini seçiyoruz.



- Port ayarlarını yaptıktan sonuçları görüntüleyebiliriz.



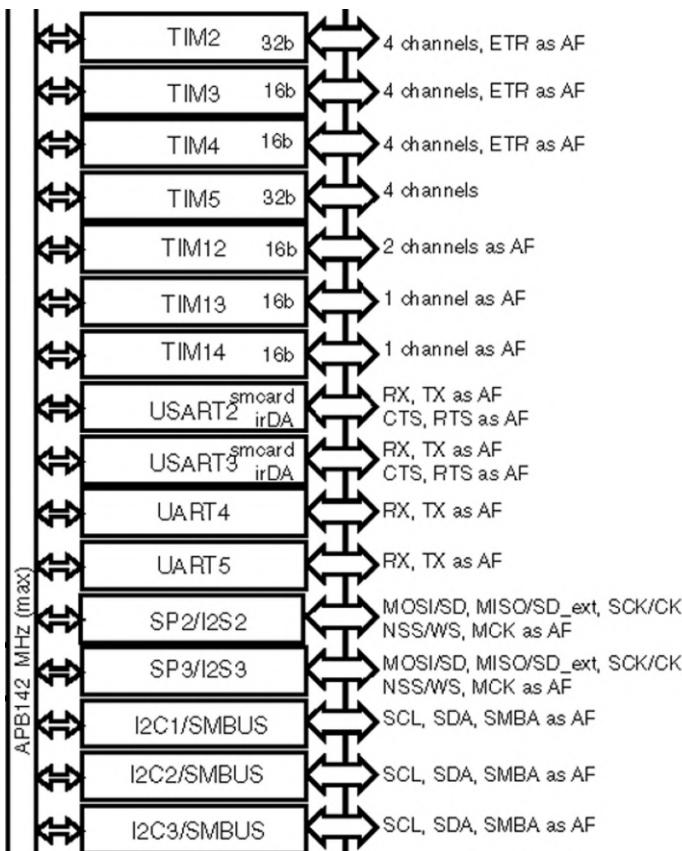
## ➤ REGISTER

### Konfigürasyon Kısımlı

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
3④ void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;          //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;    //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;    //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;    //PLLP 2
13    RCC->CR |= 0x01000000;         //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;       //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000;       //HPRE AHB 1
18    RCC->CFGR |= 0x00001400;       //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000;       //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000;        //HSERDYC
21    RCC->CIR |= 0x00800000;        //CSSC
22 }
```

- USART3 kullanacağımız. Clock hattı APB1'e gidiyor.



## RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN	
rw	rw		rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	

18.biti 1 yapıyoruz.

Bit 18 USART3EN: USART3 clock enable

Set and cleared by software.

0: USART3 clock disabled

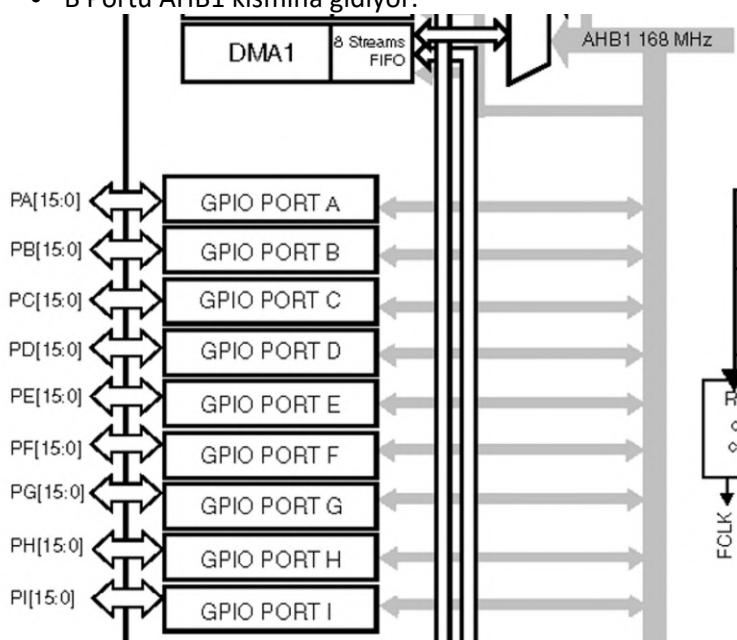
1: USART3 clock enabled

RCC->APB1ENR |= 1 << 18;

- USART3 hangi porta bağlı olduğunu öğrenmek için datasheet'e bakarız. B portun 10. ve 11.pinine bağlıymış.

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10 /11	I2C1/2/3	SPI1/SPI2/ I2S2/I2S2ext	SPI3/I2Sext /I2S3	USART1/2/3/ I2S3ext
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N	-	-	-	-
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N	-	-	-	-
	PB2	-	-	-	-	-	-	-	-
	PB3	JTDO/ TRACES WO	TIM2_CH2	-	-	-	SPI1_SCK	SPI3_SCK I2S3_CK	-
	PB4	NJTRST	-	TIM3_CH1	-	-	SPI1_MISO	SPI3_MISO	I2S3ext_SD
	PB5	-	-	TIM3_CH2	-	I2C1_SMB A	SPI1_MOSI	SPI3_MOSI I2S3_SD	-
	PB6	-	-	TIM4_CH1	-	I2C1_SCL	-	-	USART1_TX
	PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	-	USART1_RX
	PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-	-
	PB9	-	-	TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS I2S2_WS	-	-
	PB10	-	TIM2_CH3	-	-	I2C2_SCL	SPI2_SCK I2S2_CK	-	USART3_TX
	PB11	-	TIM2_CH4	-	-	I2C2_SDA	-	-	USART3_RX
	PB12	-	TIM1_BKIN	-	-	I2C2_SMB A	SPI2_NSS I2S2_WS	-	USART3_CK
	PB13	-	TIM1_CH1N	-	-	-	SPI2_SCK I2S2_CK	-	USART3_CTS
	PB14	-	TIM1_CH2N	-	TIM8_CH2N	-	SPI2_MISO	I2S2ext_SD	USART3_RTS
	PB15	RTC_REFIN	TIM1_CH3N	-	TIM8_CH3N	-	SPI2_MOSI I2S2_SD	-	-

- B Portu AHB1 kismına gidiyor.



## RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Reserved	DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved			
	rw	rw	rw	rw	rw	rw		rw	rw			rw				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved		CRCE N	Reserved			GPIOIE N	GPIOH EN	GPIOG EN	GPIOF E N	GPIOEEN	GPIOD EN	GPIOC EN	GPIOBEN	GPIOAEN		
		rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

- Biz B portunu kullandığımızdan sadece bunu aktif ediyoruz.

Bit 1 **GPIOBEN**: IO port B clock enable

This bit is set and cleared by software.

- 0: IO port B clock disabled  
1: IO port B clock enabled

RCC->[AHB1ENR](#) |= 0x00000002;

- B portun 10. ve 11. pinleri USART3 için kullanacağımızdan bu pinleri alternate function mode yapıyoruz.

## GPIO port mode register (GPIOx\_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]									
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]									
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

- 00: Input (reset state)  
01: General purpose output mode  
10: Alternate function mode  
11: Analog mode

GPIOB->[MODER](#) |= 0x00A00000;

- Kullanacağımız çevresel birim olan USART3 kullanacağımızı belirteceğiz.

## GPIO alternate function low register (GPIOx\_AFRL) (x = A..I/J/K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

## GPIO alternate function high register (GPIOx\_AFRH) (x = A..I/J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Biz 10 ve 11. pinleri kullandığımızdan high olanı kullanıyoruz.

Bits 31:0 **AFRH<sub>y</sub>**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

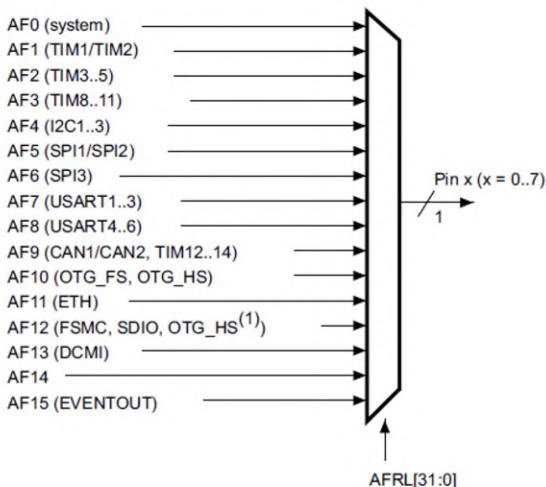
AFRH<sub>y</sub> selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

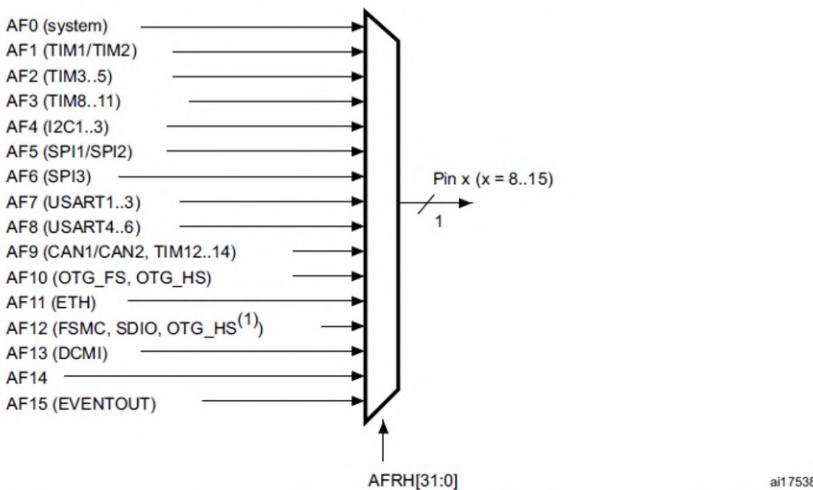
- Seçtiğimiz USART3 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitapçığına bakıyoruz.

### Selecting an alternate function

For pins 0 to 7, the GPIOx\_AFRL[31:0] register selects the dedicated alternate function



For pins 8 to 15, the GPIOx\_AFRH[31:0] register selects the dedicated alternate function



- USART3, AF7'de bulunuyor. Böylece pinlere AF7 için olan 0111 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunda belirtmemiz gerekiyor. High kullandığımızdan dizin kısmına 1 yazıyoruz.

`GPIOB->AFR[1] |= 7 << 8 | 7 << 12;`

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```
24 void GPIO_Config(void)
25 {
26     RCC->AHB1ENR |= 0x00000002;           //B clock enable
27
28     GPIOB->MODER |= 0x00A00000;          //PA10, PA11 Alternate function mode
29     GPIOB->AFR[1] |= (7 << 8) | (7 << 12); //USART3 AFRH10, AFRH11
30 }
```

- Baud rate hesabı için aşağıdaki formül kullanılır. USARTDIV ile istenen baud rate değeri hesaplanır ve bulunan değer BRR register'a yazılır.

$$\text{Tx/Rx baud} = \frac{f_{CK}}{8 \times (2 - \text{OVER8}) \times \text{USARTDIV}}$$

- OVER8 için CR1 registerinde oversampling by 16 kullandığımızdan bu değeri 0 alıyoruz.
- Baud rate ayarını 9600 yapmak için USARTDIV 273,4375 bulunur.
- Hesap yapmak yerine Reference Manual'deki tablolardan da bakılabilir.
- Biz 3.satırdağı değeri bulduk.

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 42 \text{ MHz}$			$f_{PCLK} = 84 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	1.2 KBps	1.2 KBps	2187.5	0	1.2 KBps	NA	0
2.	2.4 KBps	2.4 KBps	1093.75	0	2.4 KBps	2187.5	0
3.	9.6 KBps	9.6 KBps	273.4375	0	9.6 KBps	546.875	0
4.	19.2 KBps	19.195 KBps	136.75	0.02	19.2 KBps	273.4375	0
5.	38.4 KBps	38.391 KBps	68.375	0.02	38.391 KBps	136.75	0.02
6.	57.6 KBps	57.613 KBps	45.5625	0.02	57.613 KBps	91.125	0.02
7.	115.2 KBps	115.068 KBps	22.8125	0.11	115.226 KBps	45.5625	0.02
8.	230.4 KBps	230.769 KBps	11.375	0.16	230.137 KBps	22.8125	0.11
9.	460.8 KBps	461.538 KBps	5.6875	0.16	461.538 KBps	11.375	0.16
10.	921.6 KBps	913.043 KBps	2.875	0.93	923.076 KBps	5.6875	0.93
11.	1.792 MBps	1.826 MBps	1.4375	1.9	1.787 MBps	2.9375	0.27
12.	1.8432 MBps	1.826 MBps	1.4375	0.93	1.826 MBps	2.875	0.93
13.	3.584 MBps	N.A	N.A	N.A	3.652 MBps	1.4375	1.9
14.	3.6864 MBps	N.A	N.A	N.A	3.652 MBps	1.4375	0.93

- Ondalıklı sayının tam kısmına mantissa, küsürat kısmına fraction denir.
- Buna göre mantissa değeri 273 yani 0x111 bulunur. Geriye kalan 0.4375 değerini fraction değerine dönüştürmek için OVER8 değeri 0 olduğundan 16 ile çarpılır sonucunda 7 yani 0x7 bulunur.
- Böylece BRR registerına 0x1117 değeri yazılır.

## Baud rate register (USART\_BRR)

The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 DIV\_Mantissa[11:0]: mantissa of USARTDIV

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 DIV\_Fraction[3:0]: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV\_Fraction3 bit is not considered and must be kept cleared.

`USART3->BRR |= 0x1117;`

- USART\_BRR değerini döndürecek fonksiyon ile de yazabiliriz.

```

int BRR(double baud)
{
    double decimal = (42*1000000) / (baud*16);

    int Mantissa = (int)decimal;
    int Fraction = ceil((decimal-Mantissa)*16);

    return ((Mantissa << 4) + Fraction);
}

```

## Control register 1 (USART\_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

```
USART3->CR1 |= (1 << 2);
```

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

*Note: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.*

*When TE is set, there is a 1 bit-time delay before the transmission starts.*

```
USART3->CR1 |= (1 << 3);
```

- Mesaj geldiğinde interrupt girmesi için aktif ediyoruz.

Bit 5 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART\_SF register

```
USART3->CR1 |= (1 << 5);
```

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

```
USART3->CR1 |= (0 << 10);
```

### Bit 12 M: Word length

This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, n Stop bit

*Note: The M bit must not be modified during a data transfer (both transmission and reception)*

```
USART3->CR1 |= (0 << 12);
```

- Üsteki uyaridan sebep UE regitri en alt satırında olması gerekiyor.

### Bit 13 UE: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: USART prescaler and outputs disabled

1: USART enabled

```
USART3->CR1 |= (1 << 13);
```

## Control register 2 (USART\_CR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

### Bits 13:12 STOP: STOP bits

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

*Note: The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.*

```
USART3->CR2 |= (0 << 12);
```

- USART fonksiyonu için ayarlamalar bitti.

```
void USART_Config(void)
{
    RCC->APB1ENR |= 1 << 18; //USART3EN

    USART3->BRR |= 0x1117; //BaudRate 9600
    USART3->CR1 |= (1 << 2); //Receiver Enable
    USART3->CR1 |= (1 << 3); //Transmitter Enable
    USART3->CR1 |= (1 << 5); //RXNE Interrupt Enable
    USART3->CR1 |= (0 << 10); //Parity Control Disabled
    USART3->CR1 |= (0 << 12); //Word Length 8 Data Bits
    USART3->CR2 |= (0 << 12); //1 Stop bit
    USART3->CR1 |= (1 << 13); //USART Enable
}
```

### Kod Kısımları

- Şimdi interrupt için fonksiyon yazacağız. Öncelikle NVIC için fonksiyon yazıyoruz.

## Status register (USART\_SR)

Address offset: 0x00

Reset value: 0x0000 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

- 7.pini interrupt veriyoruz.

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART\_CR1 register. It is cleared by a write to the USART\_DR register.

0: Data is not transferred to the shift register

1: Data is transferred to the shift register

*Note: This bit is used during single buffer transmission.*

```
NVIC->ISER[1] |= (1 << 7);
```

```
49 @void NVIC_Config(void)
50 {
51     NVIC->ISER[1] |= (1 << 7);           //Interrupt Set Enable Register
52 }
```

- Şimdi Interrupt fonksiyonu yazıyoruz. Öncelikle değişken ataması yapıyoruz. Haberleşme durumunu Status Register ile bu değişkene yazacağız.
- Daha sonra gelen mesajları dizeye alıyoruz. Bunun için Data register kullanıyoruz.

## Data register (USART\_DR)

Address offset: 0x04

Reset value: 0xFFFF XXXX

Bits 31:9 Reserved, must be kept at reset value

Bits 8:0 **DR[8:0]**: Data value

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting with the parity enabled (PCE bit set to 1 in the USART\_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

```
3 char Rx_Buff[100];
4 int i=0;
```

```
54 @void USART3_IRQHandler()
55 {
56     volatile int Str;
57     Str = USART3->SR;
58     Rx_Buff[i] = USART3->DR;
59     i++;
60 }
```

```

62 void Send_Char(char message)
63 {
64     while(!(USART3->SR & 1 << 7));
65     USART3->DR = message;
66 }

68 void Send_Message(char *Str)
69 {
70     while(*Str)
71     {
72         Send_Char(*Str);
73         Str++;
74     }
75 }

77 int main(void)
78 {
79     RCC_Config();
80     GPIO_Config();
81     USART_Config();
82     NVIC_Config();
83
84     while (1)
85     {
86         Send_Message("Hello World \n");
87         for(int i=0; i<1000000; i++);
88     }
89 }
90 }
```

- Send\_Message'a mesajımızı yazıyoruz. Mesajımızdaki harflerin yani her dizideki elemanlarını alıyoruz ve her elemanı Send\_Char'a gönderiyoruz ve burada Data register'a yazarak karşı tarafa mesaj ulaşıyor

# SPI Kullanımı

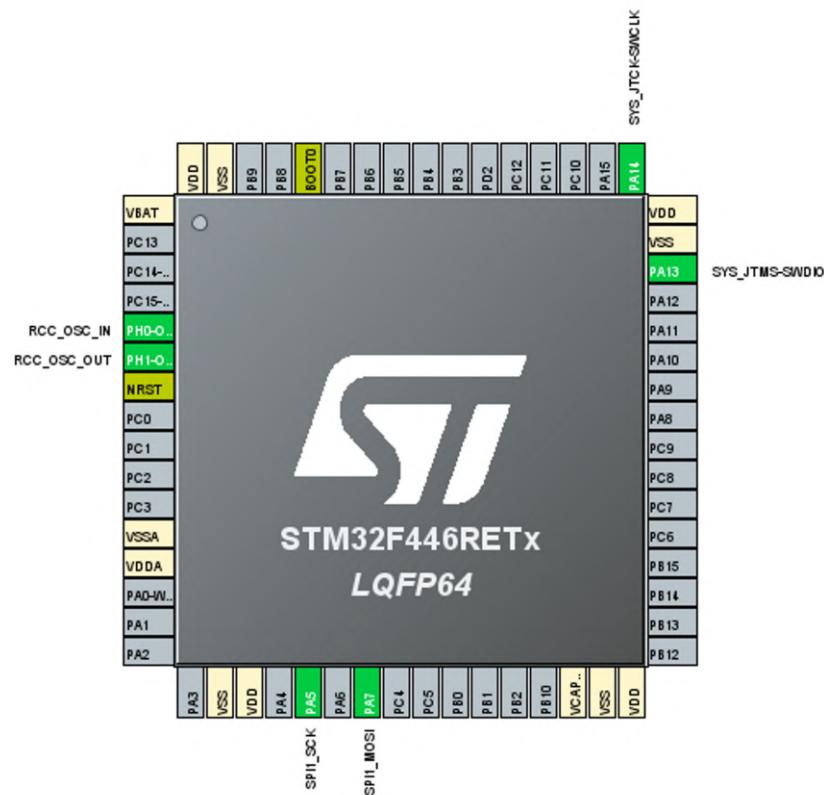
14 Mayıs 2022 Cumartesi 11:49

## SPI Kullanımı

➤ HAL

Master

Konfigürasyon Kısımlı



Pin ...	Signal on Pin	GPIO o...	GPIO m...	GPIO ...	Maximu...	User ...	Modif...
PA5	SPI1_SCK	n/a	Alternat...	No pul...	Very Hi...	<input type="checkbox"/>	
PA7	SPI1_MOSI	n/a	Alternat...	No pul...	Very Hi...	<input type="checkbox"/>	

Mode

Hardware NSS Signal

### Basic Parameters

Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First

### Clock Parameters

Prescaler (for Baud Rate)	2
Baud Rate	8.0 MBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge

### Advanced Parameters

CRC Calculation	Disabled
NSS Signal Type	Software

Kod Kısımlı

```

57 /* USER CODE BEGIN 0 */
58 uint8_t TxBuffer[1] = {0};
59 /* USER CODE END 0 */

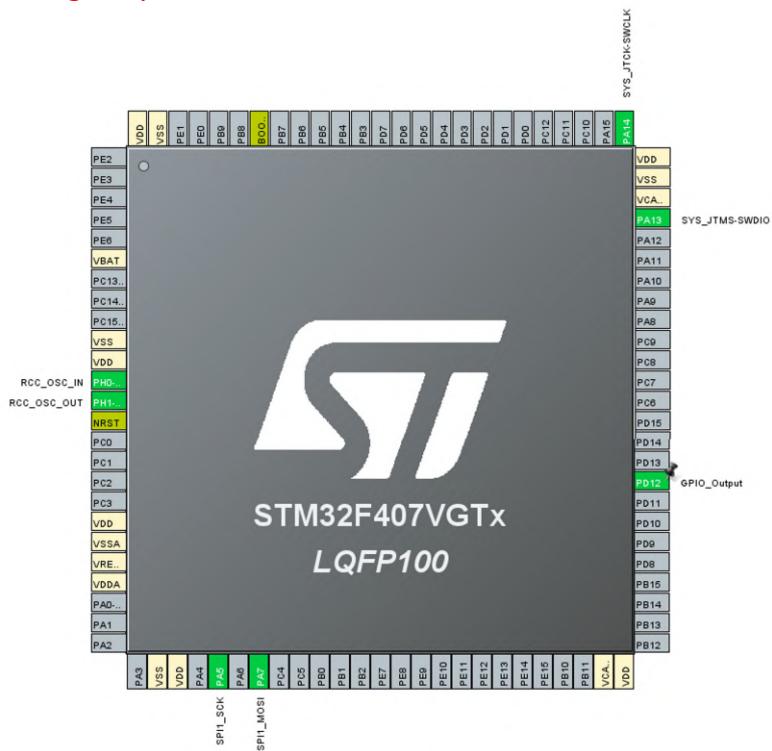
60
61 /* USER CODE BEGIN 2 */
62 int i=0;
63 /* USER CODE END 2 */

64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96 while (1)
97 {
98     /* USER CODE END WHILE */
99
100    /* USER CODE BEGIN 3 */
101    for(i= 0; i< 50; i++){
102        TxBuffer[0] = i;
103
104        HAL_SPI_Transmit (&hspi1, TxBuffer, sizeof(TxBuffer), 1000);
105        HAL_Delay(1000);
106    }
107 }
108 /* USER CODE END 3 */
109 }

```

## Slave

### Konfigürasyon Kısımları



Pin ...	Signal o...	GPIO ou...	GPIO m...	GPIO P...	Maximu...	User La...	Modifi...
PD12	n/a	Low	Output P...	No pull-u...	Low		<input type="checkbox"/>
Pin ...	Signal on Pin	GPIO ou...	GPIO m...	GPIO P...	Maximu...	User La...	Modifi...
PA5	SPI1_SCK	n/a	Alternat...	No pull-u...	Very High		<input type="checkbox"/>
PA7	SPI1_MOSI	n/a	Alternat...	No pull-u...	Very High		<input type="checkbox"/>

Mode Receive Only Slave

Hardware NSS Signal Disable

#### Basic Parameters

Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First

#### Clock Parameters

Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge

#### Advanced Parameters

CRC Calculation	Disabled
NSS Signal Type	Software

#### Kod Kismi

```
44 /* USER CODE BEGIN PV */
45 uint8_t RxBuffer[1];
46 /* USER CODE END PV */

96 while (1)
97 {
98     /* USER CODE END WHILE */
99
100    /* USER CODE BEGIN 3 */
101    HAL_SPI_Receive (&hspi1, RxBuffer, sizeof(RxBuffer), 1000);
102
103    if (RxBuffer[0] == 3)
104    {
105        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
106    }
107    else{
108        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
109    }
110    HAL_Delay(1000);
111}
112 /* USER CODE END 3 */
113 }
```

# I2C Kullanımı

25 Aralık 2021 Cumartesi 01:02

## I2C Kullanımı

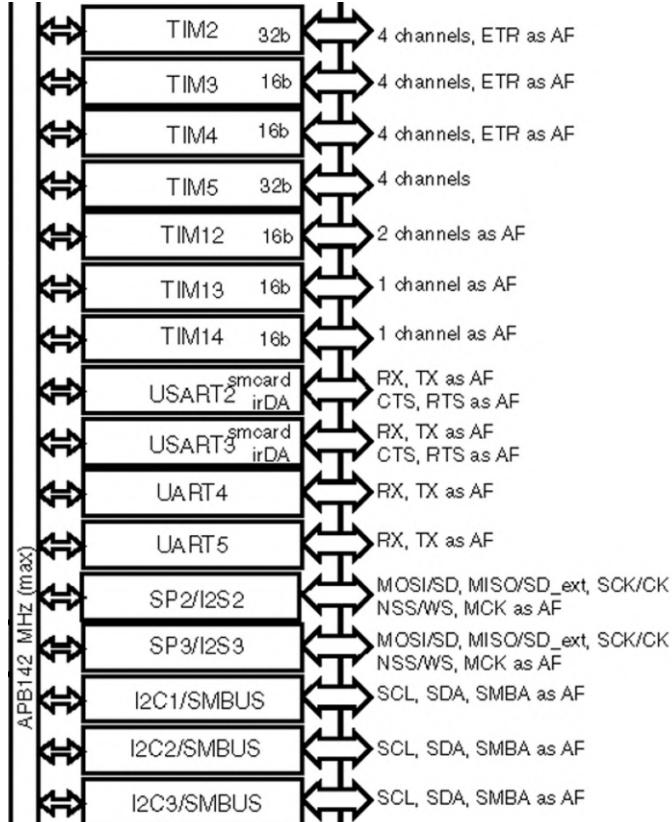
### ➤ REGISTER

#### Konfigürasyon Kısmı

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
3 @void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;          //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;    //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;    //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;    //PLLP 2
13    RCC->CR |= 0x01000000;         //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;        //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000;        //HPRE AHB 1
18    RCC->CFGR |= 0x00001400;        //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000;        //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000;         //HSERDYC
21    RCC->CIR |= 0x00800000;         //CSSC
22 }
```

- I2C2 kullanacağımız Clock hattı APB1'e gidiyor



## RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN	Reserved	
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

22.biti 1 yapıyoruz.

Bit 22 **I2C2EN**: I2C2 clock enable

This bit is set and cleared by software.

0: I2C2 clock disabled

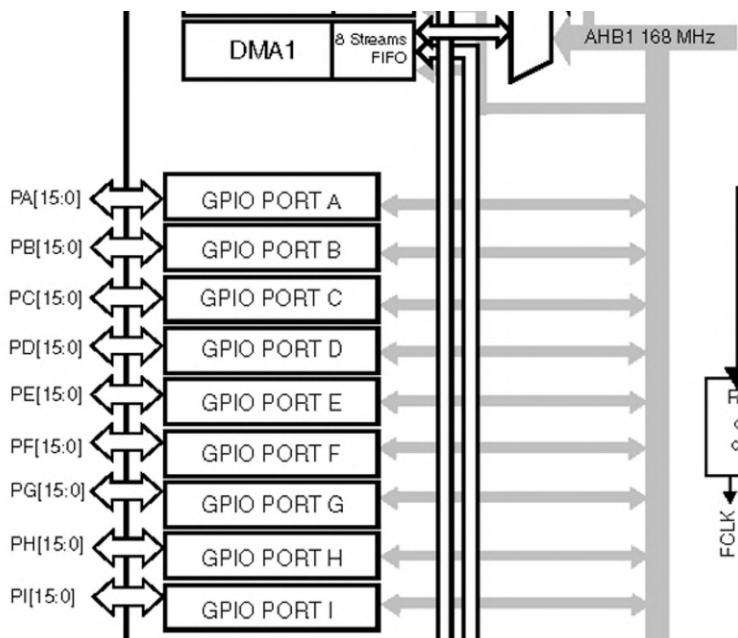
1: I2C2 clock enabled

RCC->APB1ENR |= 1 << 22;

- I2C2 hangi porta bağlı olduğunu öğrenmek için datasheet'e bakarız. B portun 10. ve 11.pinine bağlımış.

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
	SYS	TIM1/2	TIM3/4/5	TIM8/9/10 /11	I2C1/2/3	SPI1/SPI2/ I2S2/I2S2e xt	SPI3/I2Sext /I2S3	USART1/2/3/ I2S3ext
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N	-	-	-
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N	-	-	-
	PB2	-	-	-	-	-	-	-
	PB3	JTDO/ TRACES WO	TIM2_CH2	-	-	SPI1_SCK	SPI3_SCK I2S3_CK	-
	PB4	NJTRST	-	TIM3_CH1	-	SPI1_MISO	SPI3_MISO	I2S3ext_SD
	PB5	-	-	TIM3_CH2	-	SPI1莫斯I	SPI3莫斯I I2S3_SD	
	PB6	-	-	TIM4_CH1	-	I2C1_SCL	-	USART1_TX
	PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	USART1_RX
	PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-
	PB9	-	-	TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS I2S2_WS	-
	PB10	-	TIM2_CH3	-	-	I2C2_SCL	SPI2_SCK I2S2_CK	-
	PB11	-	TIM2_CH4	-	-	I2C2_SDA	-	USART3_RX
	PB12	-	TIM1_BKIN	-	-	I2C2_SMBA	SPI2_NSS I2S2_WS	-
	PB13	-	TIM1_CH1N	-	-	-	SPI2_SCK I2S2_CK	-
	PB14	-	TIM1_CH2N	-	TIM8_CH2N	-	SPI2_MISO	I2S2ext_SD
	PB15	RTC_REFIN	TIM1_CH3N	-	TIM8_CH3N	-	SPI2_MOSI I2S2_SD	USART3_RTS

- Portlar AHB1 kısmasına gidiyor.



## RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

- Buton için A portunu, I<sub>2</sub>C2 için B portunu kullandığımızdan bu portları aktif ediyoruz.

Bit 0 **GPIOAEN**: IO port A clock enable

This bit is set and cleared by software.

0: IO port A clock disabled

### 1: IO port A clock enabled

Bit 1 **GPIOBEN**: IO port B clock enable

This bit is set and cleared by software.

0: IO port B clock disabled

1: IO port B clock enabled

RCC->AHB1ENR |= 0x00000002;

- B portun 10. ve 11. pinleri I2C için kullanacağımızdan bu pinleri alternate function mode yapıyoruz.

## GPIO port mode register (GPIOx\_MODER) (x = A..I/J/K)

Address offset: 0x00

**Reset values:**

- 0xA800 0000 for port A
  - 0x0000 0280 for port B
  - 0x0000 0000 for other ports

## GPIO port mode register (GPIOx\_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 MODERy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

GPIOB->MODER |= (2 << 20) | (2 << 22) ;

- Kullanacağımız çevresel birim olan I2C kullanacağımızı belirteceğiz.

## GPIO alternate function low register (GPIOx\_AFRL) (x = A..I/J/K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

## GPIO alternate function high register (GPIOx\_AFRH)

(x = A..I/J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Biz 10 ve 11. pinleri kullandığımızdan high olanı kullanıyoruz.

Bits 31:0 **AFRH<sub>y</sub>**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

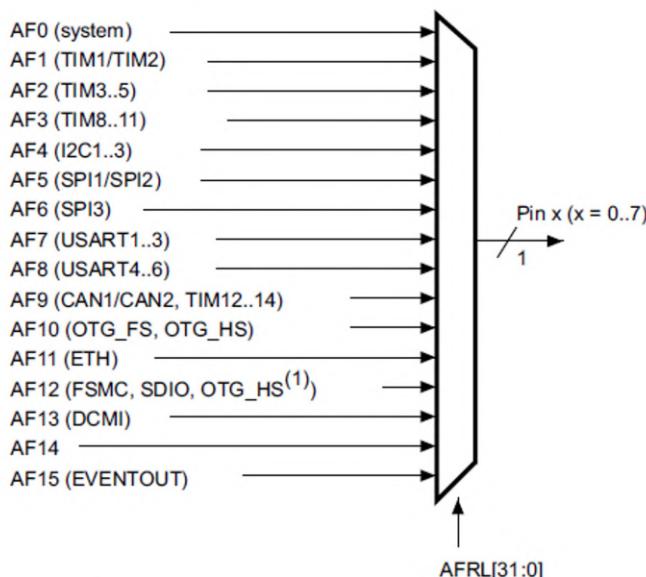
AFRH<sub>y</sub> selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

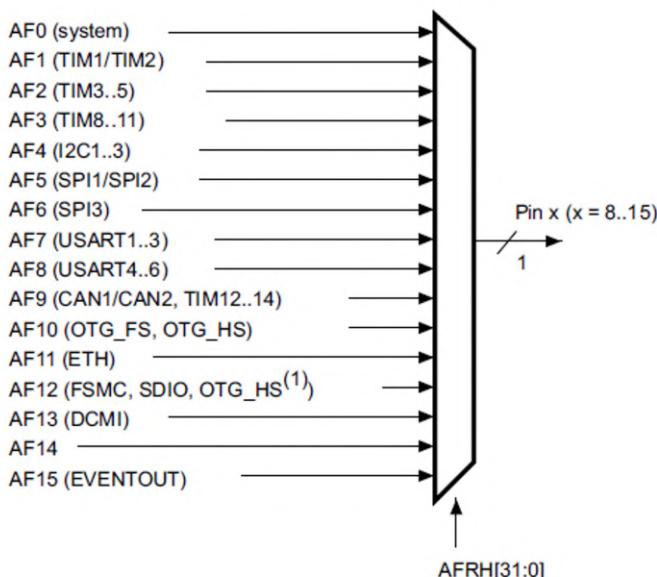
- Seçtiğimiz I<sup>2</sup>C2 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitapçığına bakıyoruz.

#### Selecting an alternate function

For pins 0 to 7, the GPIOx\_AFR[31:0] register selects the dedicated alternate function



For pins 8 to 15, the GPIOx\_AFRH[31:0] register selects the dedicated alternate function



ai17538

- I<sup>2</sup>C, AF7'de bulunuyor. Böylece pinlere AF4 için olan 0100 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunda belirtmemiz gerekiyor. High kullandığımızdan 1 yazıyoruz.

GPIOB->AFR[1] |= (4 << 8) | (4 << 12);

## GPIO port output type register (GPIOx\_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

```
GPIOB->OTYPER |= (1 << 10) | (1 << 11);
```

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```
24 void GPIO_Config()
25 {
26     RCC->AHB1ENR |= (3 << 0); //A, B clock enable
27
28     GPIOB->MODER |= (2 << 20) | (2 << 22); //PB10, PB11 Alternate function mode
29     GPIOB->AFR[1] |= (4 << 8) | (4 << 12); //I2C2 AFRH10, AFRH11
30     GPIOB->OTYPER |= (1 << 10) | (1 << 11); //Output open-drain
31 }
```

## I<sup>2</sup>C Control register 2 (I2C\_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved	FREQ[5:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 5:0 **FREQ[5:0]**: Peripheral clock frequency

The FREQ bits must be configured with the APB clock frequency value (I2C peripheral connected to APB). The FREQ field is used by the peripheral to generate data setup and hold times compliant with the I2C specifications. The minimum allowed frequency is 2 MHz, the maximum frequency is limited by the maximum APB frequency and cannot exceed 50 MHz (peripheral intrinsic maximum limit).

0b000000: Not allowed

0b000001: Not allowed

0b000010: 2 MHz

...

0b110010: 50 MHz

Higher than 0b100100: Not allowed

```
I2C2->CR2 |= 0x0008;
```

## I<sup>2</sup>C Clock control register (I2C\_CCR)

Address offset: 0x1C

Reset value: 0x0000

$f_{PCLK1}$  must be at least 2 MHz to achieve Sm mode I<sup>2</sup>C frequencies. It must be at least 4 MHz to achieve Fm mode I<sup>2</sup>C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I<sup>2</sup>C Fm mode clock.

The CCR register must be configured only when the I<sup>2</sup>C is disabled (PE = 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 11:0 CCR[11:0]: Clock control register in Fm/Sm mode (Master mode)

Controls the SCL clock in master mode.

Sm mode or SMBus:

$$T_{high} = CCR * T_{PCLK1}$$

$$T_{low} = CCR * T_{PCLK1}$$

Fm mode:

If DUTY = 0:

$$T_{high} = CCR * T_{PCLK1}$$

$$T_{low} = 2 * CCR * T_{PCLK1}$$

If DUTY = 1: (to reach 400 kHz)

$$T_{high} = 9 * CCR * T_{PCLK1}$$

$$T_{low} = 16 * CCR * T_{PCLK1}$$

For instance: in Sm mode, to generate a 100 kHz SCL frequency:

If FREQR = 08,  $T_{PCLK1} = 125$  ns so CCR must be programmed with 0x28  
(0x28 => 40d x 125 ns = 5000 ns.)

Note: The minimum allowed value is 0x04, except in FAST DUTY mode where the minimum allowed value is 0x01

$t_{high} = t_r(SCL) + t_w(SCLH)$ . See device datasheet for the definitions of parameters.

$t_{low} = t_f(SCL) + t_w(SCLL)$ . See device datasheet for the definitions of parameters.

I<sup>2</sup>C communication speed,  $f_{SCL} \sim 1/(t_{high} + t_{low})$ . The real frequency may differ due to the analog noise filter input delay.

The CCR register must be configured only when the I<sup>2</sup>C is disabled (PE = 0).

- 100 kHz çalışmak için 0x28 yazınız demiş.

I2C2->[CR2](#) |= 0x0008;

## I<sup>2</sup>C TRISE register (I2C\_TRISE)

Address offset: 0x20

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved												TRISE[5:0]				
												rw	rw	rw	rw	rw

Bits 15:6 Reserved, must be kept at reset value

Bits 5:0 TRISE[5:0]: Maximum rise time in Fm/Sm mode (Master mode)

These bits should provide the maximum duration of the SCL feedback loop in master mode.

The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration.

These bits must be programmed with the maximum SCL rise time given in the I<sup>2</sup>C bus specification, incremented by 1.

For instance: in Sm mode, the maximum allowed SCL rise time is 1000 ns.

If, in the I2C\_CR2 register, the value of FREQR[5:0] bits is equal to 0x08 and  $T_{PCLK1} = 125$  ns therefore the TRISE[5:0] bits must be programmed with 09h.

$$(1000 \text{ ns} / 125 \text{ ns} = 8 + 1)$$

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the  $t_{HIGH}$  parameter.

Note: TRISE[5:0] must be configured only when the I<sup>2</sup>C is disabled (PE = 0).

- Geri besleme döngüsünün maksimum süresi için I2C\_CR2'nin FREQR[5:0] bitlerine 0x08 yazdığımızdan

TRISE[5:0] bitlerine 0x09 yazmalıyız.

I2C2->TRISE |= 0x09;

## I<sup>2</sup>C Control register 1 (I2C\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bit 0 **PE**: Peripheral enable

0: Peripheral disable

1: Peripheral enable

*Note: If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.*

*All bit resets due to PE=0 occur at the end of the communication.*

*In master mode, this bit must not be reset before the end of the communication.*

- I2C2 aktif edildi.

I2C2->CR1 |= (1 << 0);

I2C için yazdığımız fonksiyon aşağıdaki gibidir.

```
33 void I2C_Config()
34 {
35     RCC->APB1ENR |= 1 << 22;           //I2CEN
36
37     I2C2->CR2 |= 0x0008;                //8MHz
38     I2C2->CCR |= 0x0028;                //100kHz
39     I2C2->TRISE |= 0x09;                //Maximum rise time
40     I2C2->CR1 |= (1 << 0);            //Peripheral enable
41 }
```

### Kod Kısmı

- I2C için Write ve Read fonksiyonlarını yazmaya başlıyoruz.
- I2C\_Write fonksiyonuna adress ve data olmak üzere 2 parametre yazıyoruz.

Bit 8 **START**: Start generation

This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.

In Master Mode:

0: No Start generation

1: Repeated start generation

In Slave mode:

0: No Start generation

1: Start generation when the bus is free

- Start yolluyorum.

I2C2->CR1 |= (1 << 8);

## I<sup>2</sup>C Status register 1 (I2C\_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

- Yolladığım startı kontrol ediyorum.

Bit 0 **SB**: Start bit (Master mode)

0: No Start condition

1: Start condition generated.

– Set when a Start condition generated.

– Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

- While döngüsü 1 olduğu sürece çalışır.

- SB biti 1 olana dek 0 olacağından ve döngünün çalışabilmesi için başına "!" işaretini koyarak tersliyoruz.  
0.bit 1 olduğunda döngüden çıkışacaktır.

`while(!(I2C2->SR1 & (1 << 0)));`

- Slave tarafına adresini yolluyoruz. Bizim kullandığımız cihazın adresi 0x4E'dir.

## I<sup>2</sup>C Data register (I2C\_DR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **DR[7:0]** 8-bit data register

Byte received or to be transmitted to the bus.

- Transmitter mode: Byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in DR once the transmission is started (TxEn=1)
- Receiver mode: Received byte is copied into DR (RxNe=1). A continuous transmit stream can be maintained if DR is read before the next data byte is received (RxNe=1).

*Note: In slave mode, the address is not copied into DR.*

*Write collision is not managed (DR can be written if TxEn=0).*

*If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.*

`I2C2->DR = 0x4E;`

- Slave adresin gönderimini beklememiz lazım. Bunun için kontrol yapıyoruz.

## I<sup>2</sup>C Status register 1 (I2C\_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 1 **ADDR**: Address sent (master mode)/matched (slave mode)

This bit is cleared by software reading SR1 register followed reading SR2, or by hardware when PE=0.

Address matched (Slave)

0: Address mismatched or not received.

1: Received address matched.

- Set by hardware as soon as the received slave address matched with the OAR registers content or a general call or a SMBus Device Default Address or SMBus Host or SMBus Alert is recognized. (when enabled depending on configuration).

*Note: In slave mode, it is recommended to perform the complete clearing sequence (READ SR1 then READ SR2) after ADDR is set. Refer to Figure 242.*

Address sent (Master)

0: No end of address transmission

1: End of address transmission

- For 10-bit addressing, the bit is set after the ACK of the 2nd byte.

- For 7-bit addressing, the bit is set after the ACK of the byte.

*Note: ADDR is not set after a NACK reception*

```
while(!(I2C2->SR1 & (1 << 1)));
    • Master oldu mu onu kontrol ediyoruz.
```

## I<sup>2</sup>C Status register 2 (I2C\_SR2)

Address offset: 0x18

Reset value: 0x0000

*Reading I2C\_SR2 after reading I2C\_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I2C\_SR1. Consequently, I2C\_SR2 must be read only when ADDR is found set in I2C\_SR1 or when the STOPF bit is cleared.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
									DUALF	SMB HOST	SMBDE FAULT	GEN CALL				
r	r	r	r	r	r	r	r	r	r	r	r	r	Res.	TRA	BUSY	MSL

Bit 0 **MSL**: Master/slave

- 0: Slave Mode
- 1: Master Mode

- Set by hardware as soon as the interface is in Master mode (SB=1).
- Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

```
while(!(I2C2->SR2 & (1 << 0)));
```

- TxE'nin boş olmasını bekliyoruz.

## I<sup>2</sup>C Status register 1 (I2C\_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 7 **TxE**: Data register empty (transmitters)

- 0: Data register not empty
- 1: Data register empty

- Set when DR is empty in transmission. TxE is not set during address phase.
- Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

TxE is not set if either a NACK is received, or if next byte to be transmitted is PEC (PEC=1)

*Note: TxE is not cleared by writing the first data being transmitted, or by writing data when BTF is set, as in both cases the data register is still empty.*

```
while(!(I2C2->SR2 & (1 << 7)));
```

I2C2->DR = data;

## I<sup>2</sup>C Status register 1 (I2C\_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 2 **BTF**: Byte transfer finished

- 0: Data byte transfer not done
- 1: Data byte transfer succeeded

- Set by hardware when NOSTRETCH=0 and:
  - In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RxNE=1).
  - In transmission when a new byte should be sent and DR has not been written yet (TxER=1).
- Cleared by software by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE=0.

Note: The BTF bit is not set after a NACK reception

The BTF bit is not set if next byte to be transmitted is the PEC (TRA=1 in I2C\_SR2 register and PEC=1 in I2C\_CR1 register)

```
while(!(I2C2->SR2 & (1 << 2)));
```

## I<sup>2</sup>C Control register 1 (I2C\_CR1)

Address offset: 0x00

Reset value: 0x0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SWRST		Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE		Res.	SMBUS	PE
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	

Bit 9 **STOP**: Stop generation

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.

In Master Mode:

- 0: No Stop generation.
  - 1: Stop generation after the current byte transfer or after the current Start condition is sent.
- In Slave mode:
- 0: No Stop generation.
  - 1: Release the SCL and SDA lines after the current byte transfer.

```
I2C2->CR1 |= (1 << 9);
```

- I2C için yazdığımız Write fonksiyon aşağıdaki gibidir.

```
43 void I2C_Write(uint8_t adress, uint8_t data)
44 {
45     I2C2->CR1 |= (1 << 8); //Start generation
46     while(!(I2C2->SR1 & (1 << 0))); //Start bit
47     I2C2->DR = 0x4E; //Slave adress
48     while(!(I2C2->SR1 & (1 << 1))); //Received address matched
49     while(!(I2C2->SR2 & (1 << 0))); //Master Mode
50     //I2C2->DR = adress;
51     while(!(I2C2->SR2 & (1 << 0))); //Master Mode
52     while(!(I2C2->SR2 & (1 << 7))); //Data register empty
53     I2C2->DR = data;
54     while(!(I2C2->SR2 & (1 << 2))); //Data byte transfer succeeded
55     I2C2->CR1 |= (1 << 9); //Stop generation
56 }
```

- Her butona bastığımızda cihaza sırayla adres yolluyor.

```
65 int main(void)
66 {
67     RCC_Config();
68     GPIO_Config();
69     I2C_Config();
70
71     while (1)
72     {
73         if(GPIOA->IDR & 0x00000001)
74         {
75             i++;
76             delay(6300000);
77         }
78
79         switch(i)
80         {
81             case 0:
82                 I2C_Write(m_address, 0x00);
83                 break;
84             case 1:
85                 I2C_Write(m_address, 0x01);
86                 break;
87             case 2:
88                 I2C_Write(m_address, 0x02);
89                 break;
90             case 3:
91                 I2C_Write(m_address, 0x04);
92                 break;
93             case 4:
94                 I2C_Write(m_address, 0x08);
95                 break;
96             case 5:
97                 I2C_Write(m_address, 0x10);
98                 break;
99             case 6:
100                I2C_Write(m_address, 0x20);
101                break;
102            case 7:
103                I2C_Write(m_address, 0x40);
104                break;
105            case 8:
106                I2C_Write(m_address, 0x80);
107                break;
108            default:
109                i=0;
110                break;
111        }
112    }
113 }
114 }
```