**Gömülü Sistem Nedir?**

1. ***Gömülü Sistem,*** belirli bir işlevi (fonksiyonu) gerçekleştirmek için mikrodenetleyicilerle veya mikroişlemcilerle oluşturulan bir sistemdir.
   Gömülü sistemler sadece birkaç görev gerçekleştirirler, uygulandıktan sonra bunları başka amaçlarla kullanamazsınız. Örneğin, mikrodalga fırınınızın mikroişlemcisini kullanarak film izleyemezsiniz!
2. Gömülü sistemler 3 şekilde gerçekleştirilebilir:
   a. ***Mikrodenetleyici*** ile tasarlanabilir.
   b. ***Mikroişlemci*** ile de gömülü sistem tasaranabilir ancak ***hafıza birimi*** (*fonksiyonların veyahut programın bir yere kaydedilmesi için gerekli (RAM-EEPROM))* ve ***I/O*** portlarının olması gerekiyor.
   Ancak mikrodenetleyicinin içinde zaten ***program memory*** olduğu için belleği ekstra yer kazanmak için kullanabiliriz.
   c. ***Dijital devre*** ile tasaralanabilir.
3. ***Bir CPU'yu gömülü sistemde kullanmak için***, zamanlayıcı, RAM-ROM, I/O portları, analog ve dijital dönüştürücünün olması gerekir. Bunlar yoksa CPU tek başına gömülü sistemlerde kullanılamaz.

| **Dijital Devreler** | **Mikroişlemci Tabanlı (Eskiden mikrodenetleyici yoktu bu yüzden eski kaynaklarda mikroişlemci şeklinde geçmektedir)** |
|---|---|
| Hızlıdırlar. | Bir komut, 4 farklı aşamadan geçtiği için yavaştır. |
| Esnek değildir çünkü yapılan işlemler kolayca değiştirilemez. | Esnek bir yapıya sahiptirler. |
| Şifreleme algoritmalar ve haberleşme sistemleri örnek verilebilir. | Kişisel bilgisayarlar ve cep telefonları örnek verilebilir. |

| **Mikrodenetleyici (MCU – UC - µC)** | **Mikroişlemci** |
|---|---|
| **Mikrodenetleyici**; *işlemci çekirdeği*, *bellek* ve *programlanabilir I/O birimlerini* tek bir entegre devre içerisinde barından küçük bilgisayarlardır. | Yapısında *bellek*, *I/O birimleri* ve *DMA (DMA: doğrudan adreslere erişme ve üzerinde işlem yapma imkanı sunar)* bulunan devrelere **mikroişlemci** denir. |
| Gömülü sitemlerin kalbidir. | Bilgisayar sisteminin kalbidir. |
| Güç tüketimi ve maliyeti düşüktür. | Güç tüketimi fazladır. |
| Dahili bellek ve I/O portları vardır. Bu yüzden devre küçüktür. | Bellek ve I/O portları harici olarak eklenir bu yüzden devre büyüktür. |
| Aynı anda tek bir iş yapabilirler. | Aynı anda birçok işlem yapabilir. |
| Harvard Tasarım Mimarisi kullanılır. | |
| Programda değişken türüne dikkat edilmeli. Bilgisayarlarımızdaki RAM'ler büyük olduğu için kafamıza göre int, float yazabiliyoruz. Ancak mikrodenetleyicilerde büyük RAM'ler olmadığı için küçük veri tipler seçilmeli. | |

**Mikrodenetleyicilerde Hafıza Türleri (Boyuta Göre Sıralı)**

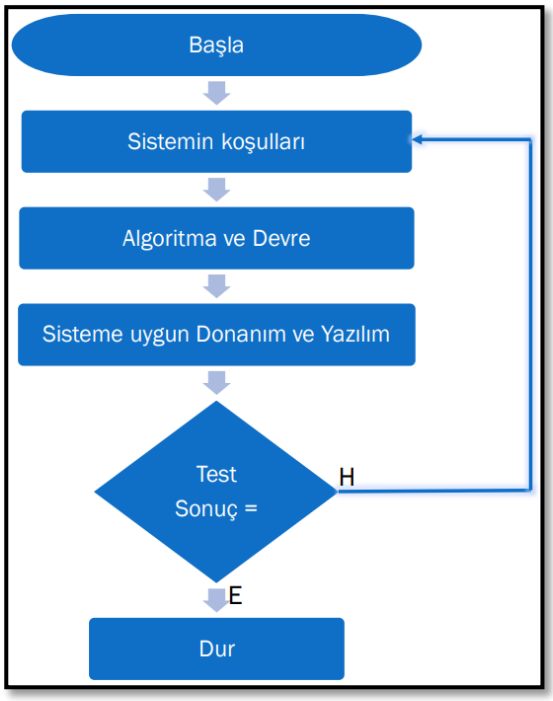
1. **Flash Memory**
   Kodun bulunduğu kısım yani sabit disk; EEPROM türündedir ve kalıcıdır.
   Mikrodenetleyici için bilgisayarlarımızdaki harddisk gibidir.
2. **Harici Disk**
3. **Main Memory (RAM)**
4. **Register**

**Yazılım**

**Mikrodenetleyiciler üç şekilde kodlanabilir:**

1. Assembly
2. Yüksek Seviyeli Dil (C, C++, Pascal)
3. Kendine has derleyici tarafından çalıştırılarak ***.hex*** uzantılı bir dosya oluşturulur.

## Gömülü Sistemlerin Tasarımı




| | AVR (Atmel) | 8051 (Intel) | PIC | HC11 (Motorola) |
|---|---|---|---|---|
| **Structure** | Harvard | Von Neumann | Harvard | Von Neumann |
| **Tech CPU** | RISC | CISC | RISC | CISC |
| **Max Frequency** | 20 MHz | 24 MHz | 20 MHz | 8 MHz |
| **Pulse / Inst** | 1 | 12 | 4 | 8 |
| **Inst / Sec** | 20 MIPS | 2 MIPS | 5 MIPS | 1 MIPS |
| **Num. Inst** | 132 | 215 | 32 | 200 |
| **Flash MM** | 256 KB | 32 KB | 64 KB | 32 KB |
| **Data Bus** | 16 | 8 | 12 | 8 |

1. **Structure,** tasarım mimarisini temsil etmektedir.
2. **Tech CPU** komut mimarisini temsil etmektedir.
3. **Max frequency,** hızı temsil etmektedir.
4. **Pulse – Instruction,** pulse'ın geçmesiyle bir komutu gerçekleştirir. Mesela 8051 için, 12 tane pulse'ın (clock) geçmesiyle 1 tane komut çalışıyor. Fakat 8051'in hızı 24 MHz diğerlerinin ise 20 MHz ve 8 MHz. Buna dikkat etmek gerekir.
5. **Instruction – Second,** bir saniyede gerçekleşen komut miktarını temsil etmektedir.
6. **Number Instruction,** kaç tane komut olduğunu gösterir.
7. **Flash Memory,** harddisk gibi düşünüleiblir.
8. **Data Bus,** veri yoludur. Dikkat ederseniz 8051 mikrodenetleyicisinin data bus'ı 8 bit dolayısıyla 8051 mikrodenetleyicisi 8 bitmiş.
9. Harvard mimarisinde komut yolu her zaman işlemci registerinin bit sayısına bağlıdır, onu geçemez. Veri yolu için eğer herhangi bir özellik (analog veya digital converter veya başka şeyler) bit sayısını aşıyorsa iki kaydedici yan yana gelmiştir tasarımı böyledir yani. Register 8 bit ise ancak gelen veri 10 bit ise 2 register'da beraber kaydedilir. Birine 8 birine 2 paylaşılır. Kalan 6 bit artık çöp yani bir şey yazılamaz.

AVR'nin, **Inst/Sec** yani saniyedeki komut işleme değeri PIC'nin 4 katı olduğu için **AVR en hızlı olanıdır.**

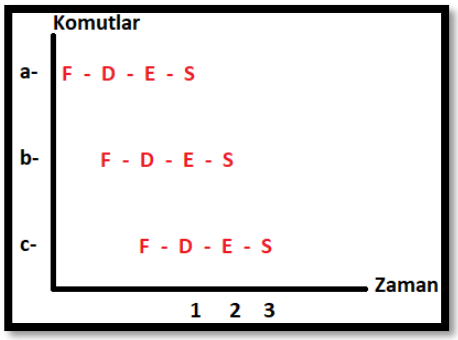
| RISC (ARM) KOMUT MİMARİSİ | CISC (X86) KOMUT MİMARİSİ |
|---|---|
| Veri türü azdır. | Veri türü fazladır. |
| Register sayısı fazla ve devresi basit olduğu için RISC daha hızlıdır.<br>Ayrıca update daha kolay çünkü devresi basit. | Register sayısı az ve devresi karmaşık olduğu için CISC daha yavaştır.<br>Ayrıca update zor çünkü devresi karmaşık. |
| Memory organizasyonunda Big Endian kullanılır. Yani MSB bitinden başlayarak yazar. | Memory organizasyonunda Little Endian kullanılır. Yani LSB bitinden başlayarak yazar. |
| Komut boyutlarına göre **fixed length modeli** kullanılır yani her komut eşit boyuttadır. | Komut boyutlarına göre **variable length modeli** kullanılır yani her komut kendi uzunluğu kadardır. |
| ***Uniform*** yapıya sahiptir yani komutların konumları sabittir. | ***Non-Uniform*** yapıdadır yani komutların konumları sabit değildir. |

**Von Neuman Tasarım Mimarisi**

- Veri yolu ve komut yolu **aynıdır,** önce komut daha sonra veri işlenir.
- Veri ve komutların genişliği aynı olmalıdır.
- Von Neuman mimarisine sahip bir bilgisayar aşağıdaki sıralı adımları gerçekleştirir:
  1. Program counter'ın gösterdiği adresten (bellekten) komut getirilir. **FETCH**
  2. Program counter'ın içeriği bir arttırılır. (Program Counter'ın arttırma özelliği vardır.)
  3. Getirilen komutun kodu kontrol birimi kullanılarak çözülür. **DECODE**
  4. İlk adıma geri dönülür.

**Harvard Tasarım Mimarisi**

- Veri yolu ile komut yolu ayrıdır dolayısıyla, verilere ve komutlara **farklı yollar**dan ulaşılır, bu sayede çalışması daha hızlıdır.
- Veri ve komutların genişliği aynı olmamalıdır.
- Harvard Tasarım Mimarisi'nde veri yolu; iletim hafıza birimleri arasında yapılıyorsa hafıza biriminin adresleme kapasitesine; iletişim CPU ve bir hafıza birimi arasında yapılıyorsa işlemcideki registerin bit sayısına bağlıdır.
- **Mikrodenetleyicilerde Harvard Tasarım Mimarisi** kullanıldığı için iki tür hafıza vardır: **Komut** ve **Değişken Hafızası.** İki tür de veri yolu vardır. Hem **komut** hem **değişkenler** için.
  Sabit (const) değişkenler program hafızasında tutulur, adresi yoktur. İşte bu yüzden sabitler (const) değiştirilemez.
- **Veri Yolu = Kaydedicinin Bit Sayısı**




| Doğru Mikrodenetleyici (UC) Seçmek | Mikroişlemci seçerken; |
|---|---|
| Bir mikrodenetleyici (MCU) seçme kriterleri:<br>1. G / Ç port sayısı.<br>2. Seri iletişim modülleri.<br>3. (Timer, ADC, PWM) gibi çevresel aygıtlar<br>4. Bellek gereksinimleri.<br>5. Gerekli işleme hızı.<br>6. Güç gereksinimleri. | 1. Bant genişliği<br>2. Bant hızına bakılmaldır.<br><br>Seri haberleşmede karşı tarafın clock'u neyse (bant genişliği) programlama tarafınında aynı olması gerekir aksi takdirde veri kaybı yaşanabilir. |

## ARDUINO MEGA 2560

Arduino Mega 2560 Teknik Özellikleri

| | |
|---|---|
| ➢ Mikrodenetleyici: **ATmega2560** | ➢ Çalışma gerilimi: **+5 V DC** |
| ➢ Tavsiye edilen besleme (çalıştırma) gerilimi: **7 - 12 V DC** | ➢ Analog giriş pinleri: **16 tane** |
| ➢ Giriş / çıkış pini başına düşen DC akım: **40 mA** | ➢ 3,3 V pini için akım: **50 mA** |
| ➢ Flash hafıza: **256 KB (8 KB bootloader için kullanılır)** | ➢ SRAM: **8 KB** |
| ➢ EEPROM : **4 KB** | ➢ Saat frekansı: **16 MHz** |
| ➢ Dijital giriş / çıkış pinleri: **54 tane (15 tanesi PWM çıkışını destekler)** | ➢ Besleme gerilimi limitleri: **6 – 20 V** |

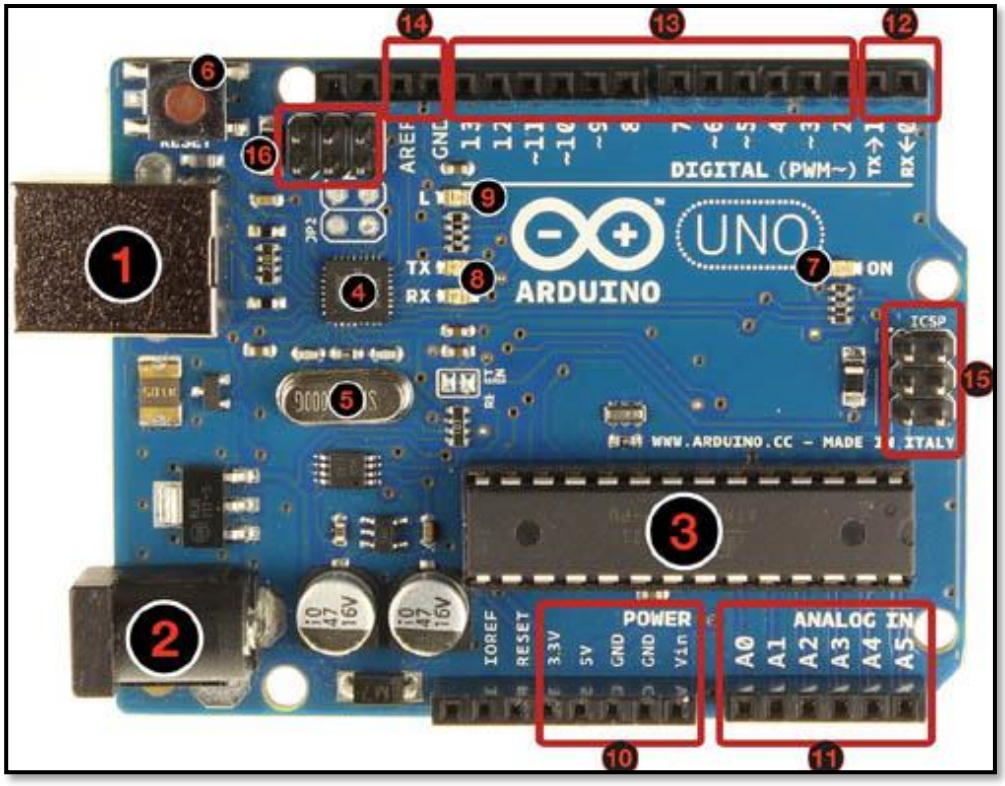
## ARDUINO UNO

Arduino UNO Teknik Özellikleri

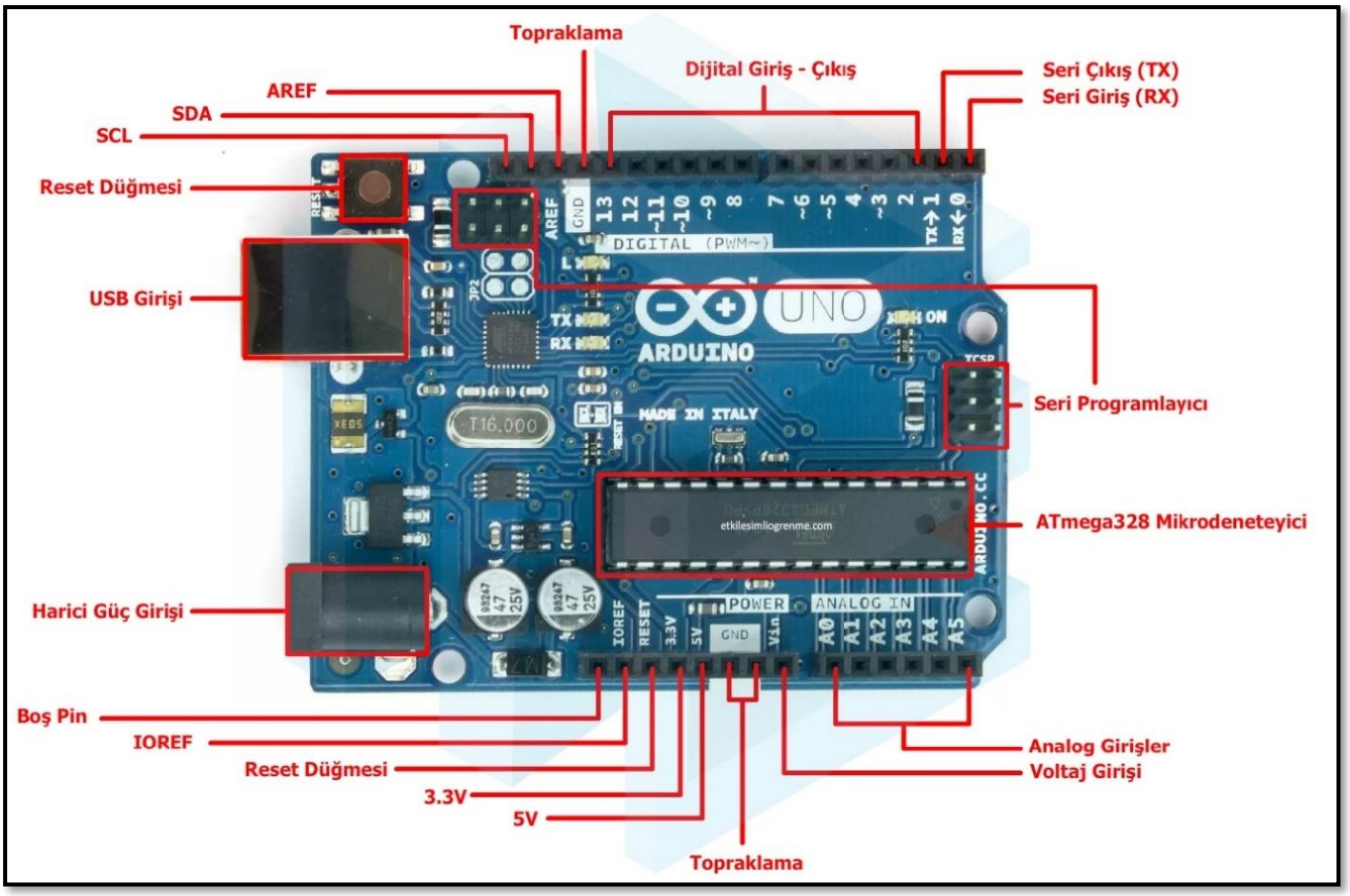
| | |
|---|---|
| ➢ Mikrodenetleyici: **ATmega328** | ➢ Çalışma gerilimi: **+5 V DC** |
| ➢ Tavsiye edilen besleme (çalıştırma) gerilimi: **7 - 12 V DC** | ➢ Analog giriş pinleri: **6 tane** |
| ➢ Giriş / çıkış pini başına düşen DC akım: **40 mA** | ➢ 3,3 V pini için akım: **50 mA** |
| ➢ Flash hafıza: **32 KB (0.5 KB bootloader için kullanılır)** | ➢ SRAM: **2 KB** |
| ➢ EEPROM: **1 KB** | ➢ Saat frekansı: **16 MHz** |
| ➢ Dijital giriş / çıkış pinleri: **14 tane (6 tanesi PWM çıkışını destekler.)** | ➢ Besleme gerilimi limitleri: **6 - 20 V** |

| | | | |
|---|---|---|---|
| **1:** USB jakı | **2:** Power jakı (7-12 V DC) | **3:** Mikrodenetleyici ATmega328 | **4:** Haberleşme çipi |
| **5:** 16 MHz kristal | **6:** Reset butonu | **7:** Power ledi | **8:** TX / NX ledleri |
| **9:** Led | **10:** Power pinleri | **11:** Analog girişler | **12:** TX / RX pinleri |
| **14:** Ground ve AREF pinleri | **15:** ATmega328 için ICSP | **16:** USB arayüzü için ICSP | |

**13:** Dijital giriş / çıkış pinleri (yanında ~ işareti olan pinler PWM çıkışı olarak kullanılabilir.)



## Arduino Üzerinde Bulunan Pinler




1. **USB Girişi (USB PLUG)**
   Bilgisayar ile Arduino arasındaki iletişimin sağlanması için Arduino üzerinde bulunan USB girişidir.

2. **Reset Düğmesi (Reset PIN)**
   Arduino içerisinde yüklü olan yazılımın resetlenmesini ve baştan başlatılmasını sağlamaktadır. Resetleme işlemi reset pininin topraklanması ile gerçekleştirilir.

3. **Volt Power PIN (3.3 V)**
   Bu pin 3.3 Voltluk bir çıkış sağlar. Devremizde 3.3 volt ile çalışması gereken bileşenler var ise beslemesi bu pinden sağlanabilmektedir. Ancak bu pinin çıkış akımı 50 mA kadardır. Bu nedenle çok yüksek akım isteyen bileşenleri bu pine bağlamamız mümkün değildir.

4. **Volt Power PIN (5 V)**
   Bu pin 5 Voltluk bir çıkış sağlamaktadır. Devremizde 5 volt ile çalışan bileşenler var ise bu pin aracılığı ile bu besleme sağlanabilmektedir. Bu pin çıkışı 450 mA'dir.

5. **Topraklama (Gnd PIN)**
   GND pin devremizin tamamlanması için gerekli topraklama pin'idir. Arduino'nun kart şeklinde tasarlanmış çeşitlerinde genellikle birden fazla Gnd PIN bulunmaktadır. Bunu nedeni Arduino GND pin üzerinde 200 mA'lik bir akım geçirebilmektedir. 2 adet GND pin ile bu değer 400 mA'e çıkmaktadır.

6. **Voltaj Girişi (Voltage In PIN)**
   Bu giriş Arduino'muzu beslemek amacı ile kullanabileceğimiz bir giriştir. Öğrenme aşamasında genellikle USB ile bilgisayara bağlı olarak kullandığımız Arduino'yu bilgisayar bağlantısı olamayan bir projede kullanmamız gerekirse, bu pin aracılığı ile Arduino'nun ihtiyacı olan gerilim sağlanabilmektedir. Bir doğru akım gerilim

kaynağının **(+) ucu Vin pinine, (-) ucu ise GND pinine** bağlanarak Arduino'muzu besleyebiliriz. Ancak üzerinde regülatör bulunmayan Arduino modellerinde (Arduino Pro Mini) bu girişe 6 Volt üzerinde bir gerilim uygulamamamız gerekmektedir. Eğer üzerinde regülatör olan bir Arduino modeli kullanıyorsak 9 Volt – 12 Volt arası bir gerilim uygulamamız gerekmektedir.

### 7. Analog Girişler (Analog In PIN)

Bu pinler analog giriş pinleridir. Çoğu Arduino modelinde 6 adet bulunmaktadır. Analog(0), Analog(1) … şeklinde isimlendirilmektedir. Bu pinler **0 – 5 Volt** arasındaki gerilimi dijitale dönüştürmektedir. Çoğu Arduino modelinin **analog giriş pin çözünürlüğü 10 bit'tir**. Bu değer 0 – 5 Volt arası uygulanan gerilim değerinin $2^{10}$ yani 1024 parçaya bölündüğünü ve bu hassasiyette (5/1024 Volt) gerilim değerinin okunabildiğini ifade etmektedir. Bu pinler analog çıkış verebilen tüm devre elemanlarının (Potansiyometre, Sensörler vb.) verdikleri değerlerin dijitale çevrilerek Arduino içerisindeki yazılm tarafından işlenebilir hale getirilmesini sağlar. Analog Giriş Pinleri aracılığı ile saniyede 10.000 değer okunabilir. Yani 10 khz hızında örnekleme yapabilirsiniz.

### 8. Djital Giriş/Çıkış (Digital I/O PINS)

Bu pinler dijital giriş / çıkış pinleridir. Elektronik bileşenlerimize Arduino içerisinde bulunan yazılıma göre 5 Volt ya da 0 Volt vermek için ya da dışarıdan 0 Volt ya da 5 Volt verildiğini algılamak için kullanılırlar. Bu Pinlerden 40 mA çıkış akımı verilebilmekte ve dışarıdan gelecek 40 mA akıma dayanabilmektedir. Arduino yazılımı içerisinde tüm dijital giriş/çıkış pinleri giriş ve çıkış olarak belirlenebilir.

### 9. PWM PINS

Bazı Dijital giriş çıkış pinlerinin yanında " ~ " işareti yer almaktadır. Bu pinler PWM (Analog Çıkış) pinleridir. Bu pinler aracılığı ile 0 – 5 Volt arası gerilim kare dalga şeklinde 8 bit çözünürlüğünde devre elemanına aktarılabilmektedir. Özetle bu pinler aracılığı ile (5/256 Volt) hassasiyetinde gerilimler üretmemiz mümkündür. Özellikle servo motor kontrolünde Arduino üzerinde bulunan PWM çıkışları yaygın olarak kullanılmaktadır.

### 10. Seri Çıkış TC ve Seri Giriş RX (Serial IN/OUT PINS)

Bu 2 pin (TX, RX) Arduinonun seri iletişim gerçekleştirmesini sağlamaktadır. Bilgisayarımız ile çoğu Arduino arasındaki iletişim, her ne kadar kullandığımız kablo, USB kablo olsa da seri port ile gerçekleştirilmektedir. Arduino üzerinde seri iletişimi USB iletişime çeviren bir çip bulunmaktadır. Bu çip bilgisayarımızdaki USB arabirimi üzerinden Arduino'nun işlemcisinin haberleşmesini sağlamaktadır. Bilgisayar dışındaki cihazlar ile iletişim kurmamız gerektiğinde (Örneğin başka bir Arduino) Serial IN/OUT pinlerini kullanmamız gerekmektedir. Ayrıca bu pinleri Arduino projelerimize Bluetooth modülü bağlayarak, Bluetooth özelliği kazandırmak içinde kullanırız.

### 11. AREF (Analog Reference PIN)

Analog Referans pini analog ölçümlerimizin doğruluğunu arttırmak için koyulmuş bir pindir. Arduino'muzu USB girişinden besliyorsak kartımız 4.8 Volt ile çalışıyor olacaktır. Analog Referans Pini boş bırakılırsa analog girişlerden alınan değerler kartın çalışma gerilimi referans alınarak dijitale dönüştürülecektir.
Örneğin 4.8 volt çıkış veren bir devre elemanı Arduino'nun analog girişine bağlandığında arduino 4.8 Volt'a karşılık 1024 değerini üretecektir. Bu da ölçümlerimizde hataya neden olacaktır. Bu nedenle daha doğru ölçümler gerçekleştirmek için Arduino Referans Pinine 5 Volt gerilim uygulanmalıdır. Referans pinin bir diğer kullanım alanı ise belirli voltaj aralığında ölçümler yaptığımız durumlarda hassasiyeti arttırmaktır. Örneğin en fazla 2.5 Volt a kadar ölçümler gerçekleştireceksek Analog Referans (AREF) pinine 2.5 voltluk bir gerilim uygulamamız bizim ölçüm hassasiyetimizin 2 kat artmasını sağlayacaktır. AREF pinine 2.5 volt uyguladığımızda Arduino'nun analog girişinin hassasiyeti 2.5/1024 olacaktır.

## Temel Devre Elemanları

### 1. Breadboard (Ekmek Tahtası)

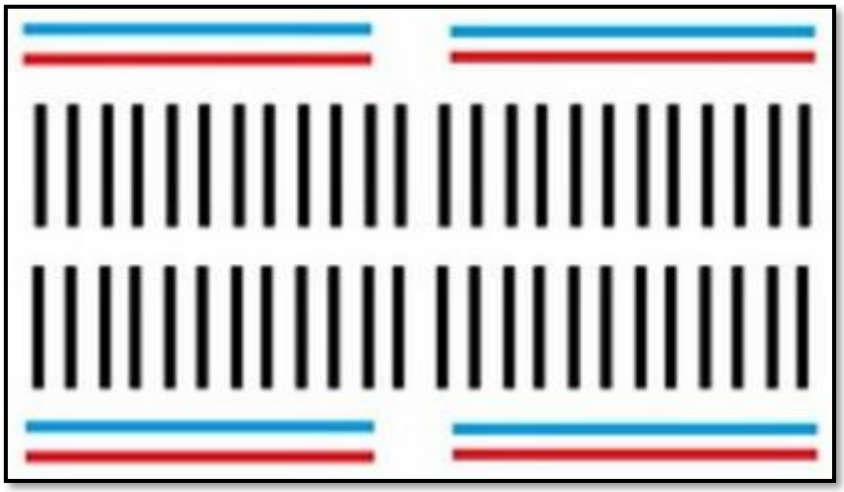
**Breadboard,** kullanacağımız elektronik elemanları bir arada tutmak ve gerekli kablo bağlantılarını gerçekleştirmek için kullanılır. Breadboard üzerinde iki çeşit yol vardır.

Güç yolları, yani beslememizin artı ve eksi uçlarını taktığımız yer, resimde görülen kırmızı ve mavi şeritlerdir.

Aşağıya doğru inen çizgilere karşılık gelen delikler kısa devre durumundadır.

Bir başka deyişle, sol üstteki kırmızıdan bağlanan bir kablo, aynı çizgi üzerinden bağlanacak kablolar ile birleşiktir. Aynı durum mavi çizgiler için de geçerlidir.

Pilin artı ucu kırmızı çizgiye, eksi ucu ise mavi çizgiye takılmalıdır.

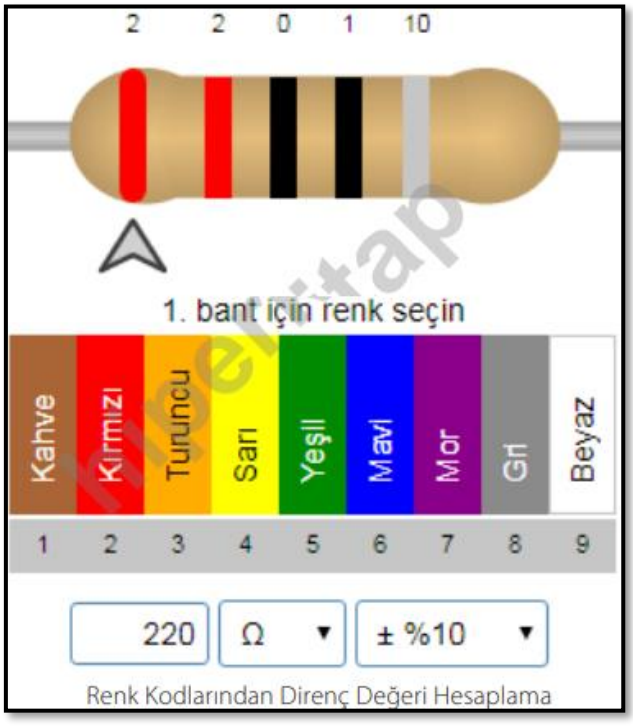


### 2. Direnç

Hat üzerinden geçen akımı ayarlamak için kullanılır.

V = İ * R formülüne sahiptir.

Örneğin, LED dediğimiz lambaların üzerinden fazla akım geçmesi bu lambalara zarar vermektedir. Bu lambaların fazla akım çekmesini engellemek için LED'in bağlantısından önce 220 Ohm değerinde bir direnç takılır. Böylece LED üzerinden geçen akım azaltılmış olur. Eğer 220 Ohm yerine daha büyük bir direnç bağlanırsa LED'in parlaklığında azalma olduğu görülür.Direncin değerinin anlaşılması için, direnç üzerinde renkli şeritler bulunur. Aşağıda "5 bantlı bir direncin renklerini" öğendik. 220 Ohm olan direnç bu renklere sahip olmaktadır.

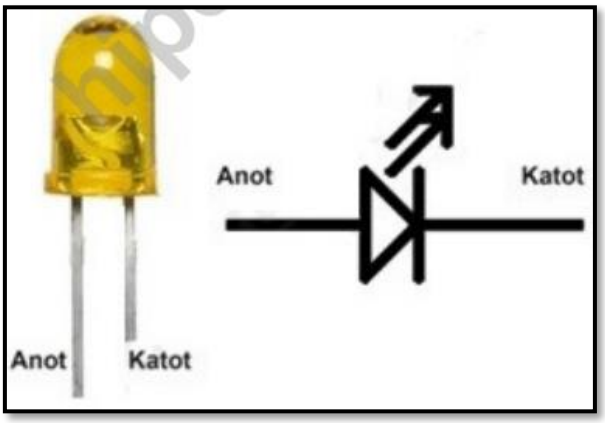


### 3. Jumper Kablo

Bunlara kısaca bir çeşit bağlantı kabloları diyebiliriz. Breadbord ve Arduino arasında bağlantı kurmak için oldukça kullanışlıdır. Uçlarında erkek ve dişi girişlerin bulunmasına göre 3 çeşit jumper kablo bulunmaktadır.


### 4. Led

Alışık olduğumuz 6V ile çalışan ufak ampullerin aksine LED'lerin anot ve katot olmak üzere iki farklı bacağı vardır. Bunlardan anot, pozitif gerilime, yani + uca; katot ise negatif gerilime, yani – uca ya da toprak hattına (GND, Ground) bağlanmalıdır.




### 5. Gerilim, Akım ve Ohm Yasası

Çeşitli devre elemanlarının farklı gerilim, yani voltajlarda çalıştığını biliyoruz. Arduino kartımız ise 5V gerilimle çalışmaktadır. 5V değeri bize kartın çıkış gerilimini ifade etmektedir.

LED'imiz için ise bu durum biraz farklıdır.LED'in üzerinden geçecek maksimum akımın 20 mA (miliamper = amperin 1000'de 1'i) değerini geçmemesi gereklidir.

LED, 20 mA akıma ihtiyaç duymakta. Eğer LED'imizi Arduino'ya doğrudan bağlayacak olursak LED üzerinden kartın sağlayabileceği maksimum değerde akım geçecek ve LED'imiz veya kartımız bozulacaktır. Bunun için akım sınırlayıcı bir direnci LED'imize seri olarak bağlamamız gerekmekte. Peki bu direncin değeri nasıl belirlenecek? İşte burada Ohm Kanunu dediğimiz denklem devreye giriyor: **V = i x R** Bu denklemde **V, bize gerilimi**; **i, akımı** ve **R ise direnci** temsil ediyor.

Eğer 20 mA akıma ihtiyaç duyan LED'i, Arduino'muzun 5V çıkış sağlayan pinlerinden birine bağlayacak olursak; **5V = 0,020A x R** denklemini elde etmiş oluruz. Bu denklemden R'yi çekecek olursak sonucu 250 buluruz. Bu demek oluyor ki LED'imizi 5V gerilimle kullanmak için 250 Ω (Ohm) değerinde bir dirence ihtiyacımız var. Tam değeri doğru tutturmamız çok önemli değil, elimizde mevcut olan 220 Ω'luk direnci kullanabiliriz.

## Ardunio Programlama Dili

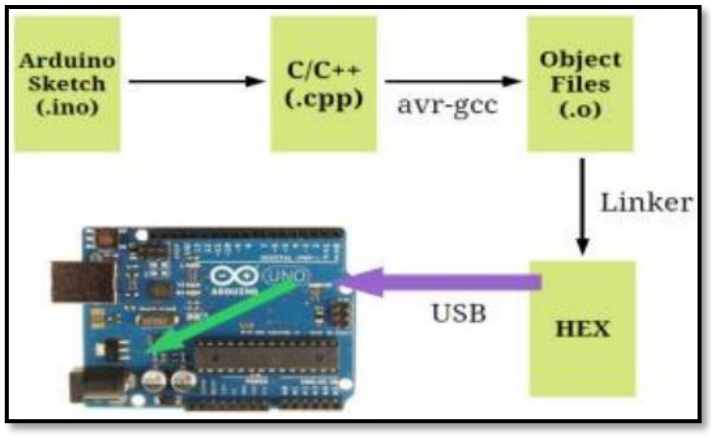
- **Arduino,** üzerinde bulundurduğu özel giriş ve çıkış portları yardımıyla, programcının yazdığı özel kodları fiziksel etkiye çeviren elektronik devre kartıdır.
- Programcı tarafından yazılan bu kodların işlenmesi için Arduino üzerinde mikroişlemciler bulunmaktadır. Bu mikroişlemcilerin türüne göre de **Arduino türleri** belli olmaktadır.
- Büyük-küçük harfe duyarlıdır.
- Programlama dilinin anahtar kelimeleri, isim olarak kullanılamaz.
- **Global değişkenler,** bütün program boyunca tanımlıdırlar, yok edilemezler.
  **Otomatik değişkenler**, tanımlı oldukları blok boyunca yaşayıp bloktan çıkınca yok edilirler.
- Ardunio yapısında **main()** fonksiyonu bulunmaz onun yerine iki temel bileşen vardır bunlar; **setup()** ve **loop()**'tur.
  - **Setup** içinde; pin ayarları yapılır. Pinlerden hem input hem output ve farklı özellikleri olduğu için bu ayarları pin'e bildirmek gerekiyor.
  - Eğer sürekli olarak değil de sadece bir kez çalışıp duracak bir fonksiyon var ise bu fonksiyon **setup** içine yazılır.
  - **Loop** yapısı sürekli olarak çalışacaktır. Bu yüzden fonksiyon bunun içine yazılırsa fonksiyon hep çalışacaktır.

## Processing & Wiring Dillleri

- **Processing dili,** görsel odaklı geliştirilen basit seviyede bir programlama dili ve ortamıdır.
  Ekstra hiç bir komut kullanmadan sadece hazır fonksiyon ve nesneleri kullanarak program yazabiliyorsunuz.
- **Wiring,** tek bir MCU bordu + bir IDE + bir programlama dilinin birleşiminden oluşan açık kaynak kodlu elektronik prototip platformudur.

## Arduino IDE Programı

- Arduino, Wiring tabanlı C/C++'a çok yakın bir dil ile programlanıyor.
  Arduino kaynak kod dosyalarına **(*.ino) Sketch (taslak)** adı veriliyor.

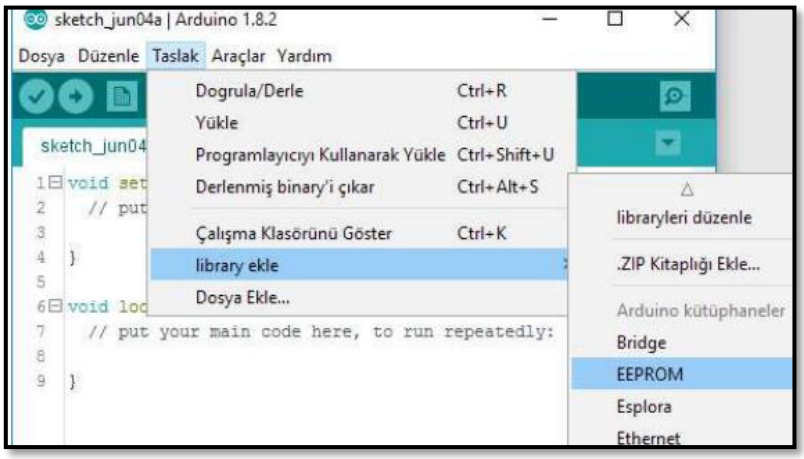



- Kontrol kartlarının üzerinde **mikrodenetleyici** bulunur. Bu mikrodenetleyicinin üstündeki bilgiye bakılarak ona göre kodlama yapılır. İşletim sistemi destekleyen işlemcilerde **.exe** dosyası üretilir.
  Arduino'da yazdığımız koddan, **GCC derleyici** de kullanılarak bir **obje(.o)** dosyası üretilir. Eğer program içinde bir kütüphane kullanıldıysa, **.o** uzantılı dosya **Linker** da kullanılarak diğer dosyalarla bağlanır ve bunun sonucunda **.hex** dosyası üretilir **(mikrodenetleyicinin linker üzerinden ürettiği dosya uzantısı .hex'dir .exe gibi düşünülebilir. Yani çift tıklayınca çalıştırılacak olan dosya)**.
  Mikrodenetleyicinin memory organizasyonunu Arduino kodu Assembly tarafından yapılır.
  **Assembly'ye çevrilmesinin sebebi** hem kompakt hem de hızlı bir şekilde kod üretmesidir. Ancak daha karmaşık programları geliştirmek için pratik ve verimli değildir.

- Bir sıcaklık kontrol devresi tasarımı yapalım;
  Öncelikle sıcaklığı algılayacağımız sensörü ve kontrol etmemiz gereken fan, ısıtıcı, soğutucu gibi elemanları belirleriz. Ardından kullanıcı ile etkileşim halinde olunacaksa, butonlar, LCD ekran gibi devre elemanlarının olup olmadığı belirlenir. Elde edilen verinin saklanması gerekiyorsa SD kart, verinin dışarı aktarılması ya da sistemin dışarıdan yönetilmesi gerektiği durumlarda ethernet/wireless gibi yöntemler kullanılabilir. Sıcaklığın arttığı/azaldığı durumlarda kullanıcıları sesli/ışıklı uyarabilmek için elemanlar da kullanılır.
  Tüm bu çevre elemanları, mikrodenetleyici/Arduino kartı seçimimizi etkiler. Basit bir algoritma ile ölçülen sıcaklığın belirli bir sıcaklık değerinin üstünde olup olmadığı kontrol edilir. Eğer istenilen sıcaklığın üstünde ise fan, soğutucu çalıştırılabilir, kullanıcı sesli ve görsel olarak uyarılır. Sıcaklık sınır değerin üstünde değilse kullanıcı yeşil led ile herşeyin normal olduğu yönünde uyarılır.
- Sıcaklığı sensör aracılığıyla ölç.
- Sıcaklığı kontrol et, Eğer sıcaklık 45 derece ve üzerinde ise sesle uyar, kırmızı led'i aç.
- Eğer değilse yeşil led'i aç.
- Önceki adımları döngü içinde tekrar et.

## Arduino Kütüphanesi ve Diğer Kütüphanelerin Kullanımı

- **SQL Server'a** bağlanabilmek için .net içerisinde hazır olarak kullanılan kütüphanelerden **System.Data.SqlClient** kütüphanesini kullanırız. Arduino içerisinde **EEprom** kullanabilmek için hazır olarak bulunan kütüphaneyi kullanabilmek için **#include <EEPROM.h>** komutu kullanılır.
- Aşağıda, hazır kütüphaneleri kullanmak için import library'den istediğiniz kütüphaneye erişme imkanı gösterilmiştir.




- **EEPROM:** EEPROM, belleklere erişimi sağlayan sınıf ve metodları içeren bir kütüphanedir. Arduino'nun enerjisi kesildiğinde de verilerin kaybı istenmediği durumlarda kullanılır.
- **Ethernet:** Projenizi internete bağlanabilmesi için kullanılan bir kütüphanedir.
- **Firmata:** Seri iletişim için kullanılan kütüphanedir.
- **SD:** Sd kartlara erişimi sağlayan okuma/yazma işlemlerini gerçekleştirebildiğiniz kütüphanedir.
- **Servo:** Servo motorları kontrol etmek için kullanılır.

## Serial Monitörün Kullanılması

- Arduino ile geliştireceğimiz projelerde pc'ye veri göndermek ya da pc'den veri almak gerektiğinde birçok arduino kartta bulunan **uart/ttl dönüştürücü** entegresi tarafından iletişim sağlanır.
- Temel olarak seri iletişim, **bus** denen hat üzerinden veri elemanları gönderilir ya da alınır. Bus hattından veriler byte byte gönderilir. Bir veri gönderilirken diğer veri hazırlanır ve gönderilir. Paralel iletişimde veri hattı 1'değil 8 tane olduğundan **aynı anda 8 veri birden** gönderilir.
- **Baud Rate,** gönderilen ve alınan verinin hızını ifade eder. Normal şartlarda aynı hızda olması gerekir. Veri gönderme/alma hızları arasında fark olursa yani veri gönderen cihaz, veriyi alan cihazdan daha hızlı ya da daha yavaş olursa iletişim esnasında veri kaybı olabilir. Seri iletişime başlamadan önce baud hızı ayarlanmalıdır. *1 baud; Her saniyede 1 karakterin gönderileceği anlamına gelir.*

## C'de Değişkenlerin Ömrü

- Programınızın tüm bölümlerinde bir değişkeni kullanmak istiyorsanız yapmanız gereken değişkeni kodlarınızın en üst kısmında tanımlamaktır. Böylece bu değişkene kodunuzun her kısmından ulaşabilirsiniz. Bu şekilde tanımlanan değişkenlere **global değişken** denmektedir.
- Otomatik değişkenler tanımlı oldukları blok boyunca yaşayıp bloktan çıkınca yok edilirler.
- pin değişkeni **setup()** metodu dışında geçerli olmadığı için pin değişkeninin, **loop()** metodu içinde tanımlanmadığına dair hata alırız.

```
void setup(){
        int pin = 13;
        pinMode(pin, OUTPUT);
        digitalWrite(pin, HIGH);
}
void loop(){
        digitalWrite(pin, LOW); //hata: pin degiskeni burada gecerli degil.
        // Iste bu gibi durumlarda Global degiskenler kullanmaliyiz.
}
```

## Matematiksel Fonksiyonlar

| Fonksiyon | Parametre | İşlem |
|---|---|---|
| pow(x, n) | x: sayı<br>n: üs değeri | Üs alma |
| sqrt(x) | x: sayı | Karekök alma |
| abs(x) | x: sayı | Mutlak değer |
| sin(x) | x: açı (radyan) | Sinüs |
| cos(x) | x: açı (radyan) | Kosinüs |
| tan(x) | x: açı (radyan) | Tanjant |
| a = min(x, y) | x: sayı-1<br>y: sayı-2 | İki sayının en küçüğü |
| a = max(x, y) | x: sayı-1<br>y: sayı-2 | İki sayının en büyüğü |
| a = constrain(x, min, max) | x: ondalık sayı | En küçük tam sayıya yuvarlama |
| a = ceil(x) | x: ondalık sayı | En büyük tam sayıya yuvarlama |

## Veri Türleri

| Veri Türü | Açıklama | Bellekteki Boyutu |
|---|---|---|
| byte | 0 ila 255 arasındaki işaretsiz (pozitif ) sayıları saklar. | 8 bit=1 byte |
| int | 2 byte'lık tam sayıları saklar. | 2 byte |
| word | 2 byte'lık işaretsiz sayıları saklar. | 2 byte |
| float | 4 byte'lık işaretli ondalıklı sayıları saklamak için kullanılır. | 4 byte |
| double | float türünden daha hassasiyet gerektiren durumlarda kullanılır. | 4 byte |
| array | Herbirine indeks numarası ile ulaşılabilen aynı türdeki verileri kapsayan bir veri katarıdır. | Dizinin veri türüne göre boyutu değişkendir. |
| string | char veri türünden oluşan karakterler bütünüdür | Her elemanı kadar 1 byte değer alır. |
| string | string veri türündeki verilerle çalışmak için kullanılan sınıftır | |
| boolean | Mantıksal veri türüdür. True ya da false olarak iki değeri vardır. | 1 byte |
| char | Tek bir karakteri barındıran veri türüdür. | 1 byte |

1. BYTE

byte 8 bitlik işaretsiz sayıları saklamak için kullanılır. 0 ila 255 arasındaki sayılar saklanır.

```
byte b = B10010; // "B" binary formati (B10010 = 18 decimal)
byte c = 45;     // decimal formatta deger verme
```

2. WORD

Word 16 bitlik pozitif tam sayıları saklamak için kullanılır. 0 ila 65535 arasındaki sayıları temsil edebilir.

```
word w = 1000;
```

3. FLOAT

```
float sayiFloat, z, sensorDeger = 1.117;
int x=1, y;
y = x / 2;          // y'nin degeri su anda 0, int sayilar kesirleri saklamaz.
z = (float)x / 2.0; // z'nin degeri .5 ( 2.0 olarak kullanmalisinz).
```

4. CHAR - STRING

Tek tırnak ile karakter temsil edilir. char örnek = 'A'; şeklinde gösterilir. Birden fazla karakteri saklamak için string veri türü kullanılır ve çift tırnak ile gösterilir.

```
char str1[15]; //15 karakterlik string tanimlar
```

str1 tanımlanırken dizi değerleri verilmedi.

```
char str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
//8 karakterli diziyi tanimlar deger verir
```

str2 dizi tanımlanırken değerleri verildi, fakat son karakterin değerini girmediğimiz için otomatik olarak dizinin sonuna yani 8. indise compiler null karakter ekledi.

```
char str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
```

str3 dizi tanımlanırken, açık olarak null karakter eklendi.

```
char str4[ ] = "arduino";
```

str4 tanımlanırken tırnak içindeki, string sabitlere ve null karaktere göre dizinin boyutu otomatik olarak derleyici tarafından ayarlandı.

```
char str5[8] = "arduino";
```

str5 tanımlanırken dizinin boyutu ve elemanları belirtildi.

```
char str6[15] = "arduino";
```

str6 tanımlanırken boyutu ve elemanları verildi fakat, daha fazla eleman alacak kadar boşluk hala mevcut

Karakter dizileriyle yapamayacağımız bir çok string işlemini String sınıfının metodlarıyla yapabiliriz. string'leri birleştirmek, birbirine eklemek, arama yapmak ve daha fazlası bu metodlarla yapılır.
Karakter dizisi olan stringler **küçük "s"** ile yazılırken String sınıfını **büyük "S"** ile başlayan tanımlamalarla kullanırız.

```
String string1 = "Bu yazi bir string'dir."; // Sabit String kullanimi
```

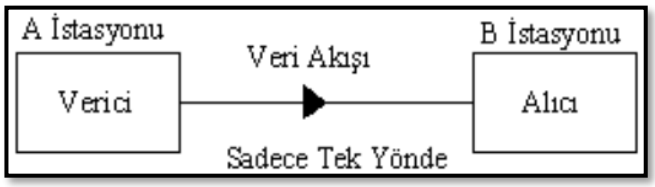
Bir metoddan dönen sonucu da bir string'e ekleyebiliriz. Aşağıda int türü döndüren analogRead() metodu bir stringe ekleniyor.

```
string2 = string1 + analogRead(A0);
```

## Veri İletim Türleri

### 1. Simplex

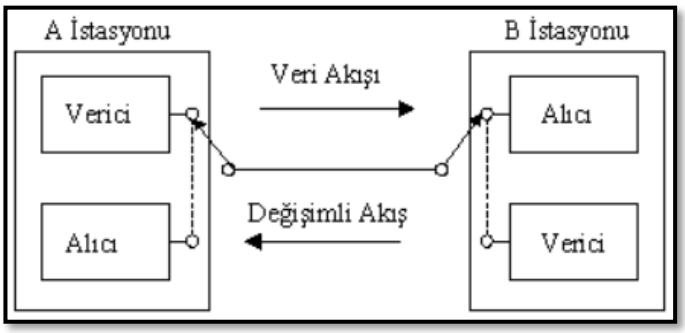
İletişim sadece tek yönlü mümkündür. Örneğin, radyo yayını. Radyo istasyonu yayın programını iletir ancak radyo alıcınızdan geri hiçbir sinyal almaz.




### 2. Half Duplex (Yarım Çift Yönlü)

İletim her iki yönde de mümkün olmasına rağmen aynı anda(bir clock'ta) sadece TX ya da RX yani veri almak ya da göndermek mümkündür. Örneğin polis telsizi Telsizde aynı anda konuşulmaz. Birisi konuşurken diğeri dinler.

Burada, "A" istasyonundaki verici, "B" istasyonundaki alıcıya veri gönderir. Ters yönde iletimde, "B" istasyonu verici konumuna dönüşür ve "A" istasyonundaki alıcı ile haberleşir.




### 3. Full Duplex (Tam Çift Yönlü)

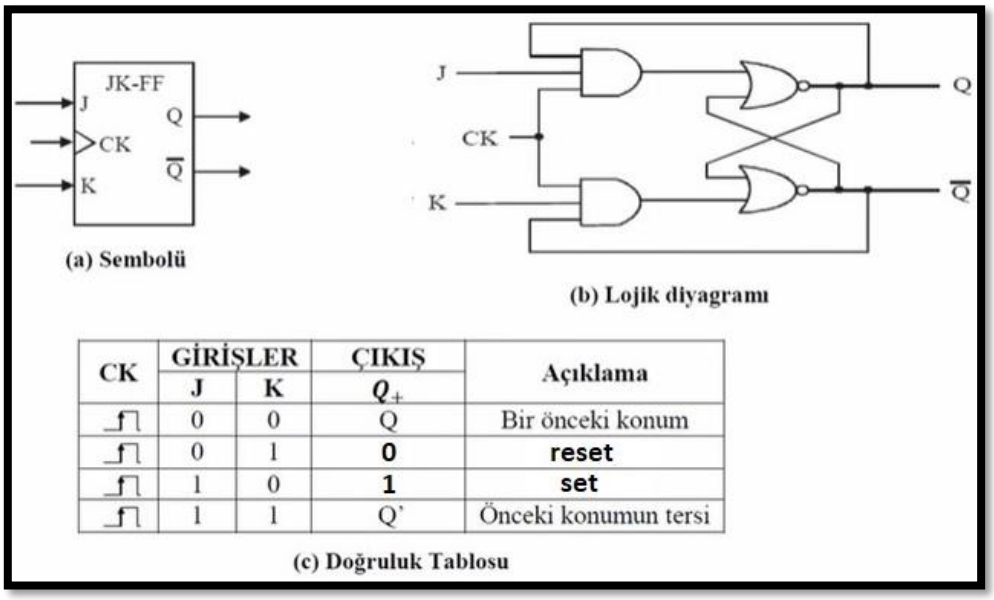
İletişim her iki yönede mümkündür, her iki yönde de aynı zamanda iletilebilir ve alınabilir. Örneğin telefon konuşması. İki taraf da aynı anda konuşup dinleyebilmektedir.

**JK Flip Flop**

**J; set,**
**K; reset yani silme anlamındadır.**

**Burada J ve K girişi, Q ise çıkışı temsil etmektedir.**

$Q_+ = J.Q' + K'.Q$




| CK | GİRİŞLER | | ÇIKIŞ | Açıklama |
|---|---|---|---|---|
| | J | K | $Q_+$ | |
| ⎍ | 0 | 0 | Q | Bir önceki konum |
| ⎍ | 0 | 1 | **0** | **reset** |
| ⎍ | 1 | 0 | **1** | **set** |
| ⎍ | 1 | 1 | Q' | Önceki konumun tersi |

(c) Doğruluk Tablosu

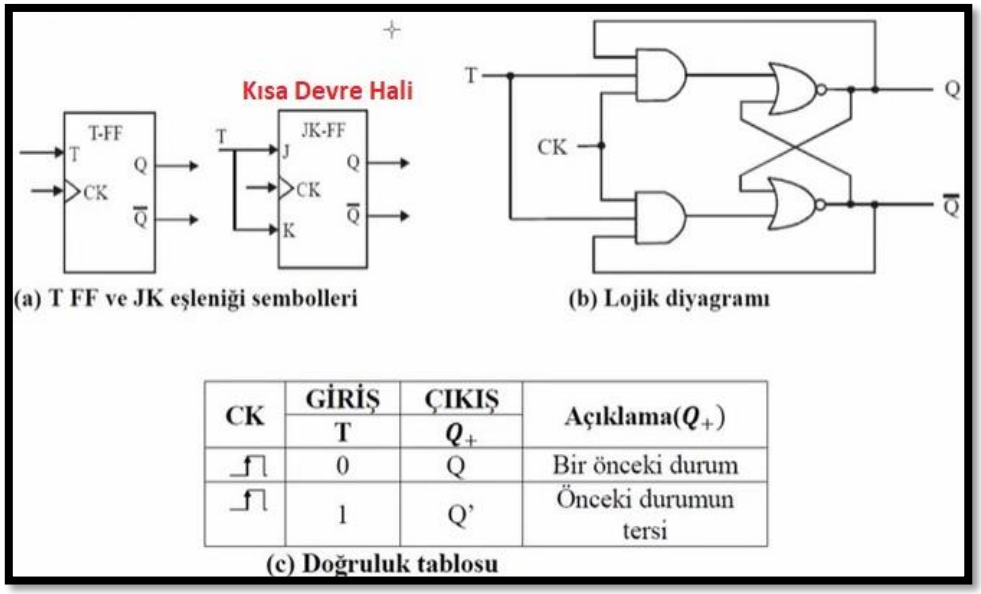
**T Flip Flop**

**T tipi Flip Flop, JK tipi Flip Flopun girişleri kısa devre yapılarak elde edilir.**

**Tek girişe sahiptirler. Burada T girişi, Q ise çıkışı temsil etmektedir.**

$Q_+ = T \oplus Q$

$Q_+ = \overline{T} \cdot Q + T \cdot \overline{Q}$




| CK | GİRİŞ | ÇIKIŞ | Açıklama($Q_+$) |
|---|---|---|---|
| | T | $Q_+$ | |
| ⎍ | 0 | Q | Bir önceki durum |
| ⎍ | 1 | Q' | Önceki durumun tersi |

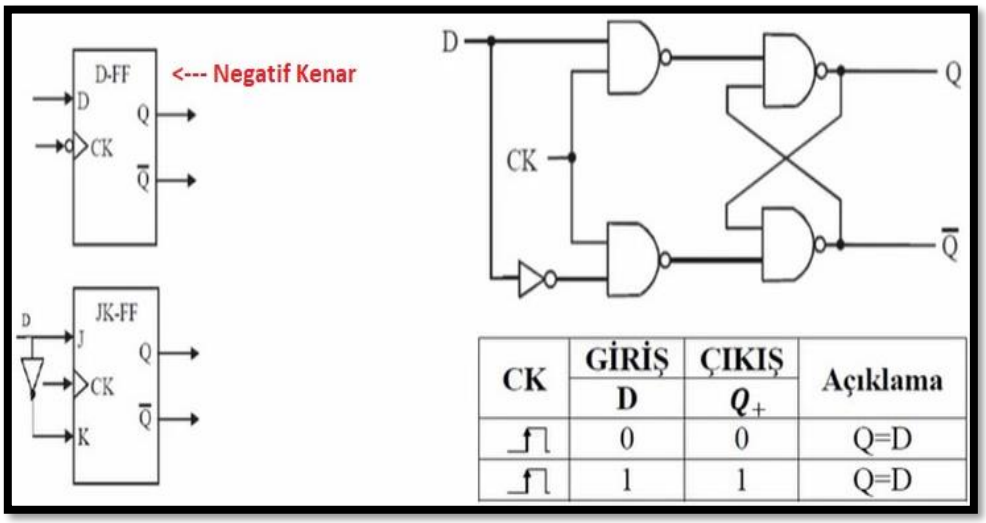
(c) Doğruluk tablosu

**D Flip Flop**

**JK tipi Flip Flopa DEĞİL (NOT) kapısı eklenip girişleri birleştirilerek elde edilir.**

**Tek girişe sahiptir. Giriş ne ise çıkışta aynısıdır.**

$Q_+ = D$




| CK | GİRİŞ | ÇIKIŞ | Açıklama |
|---|---|---|---|
| | D | $Q_+$ | |
| ⎍ | 0 | 0 | Q=D |
| ⎍ | 1 | 1 | Q=D |

**Veri İletim Yöntemleri**

Arduino kartlarda **RX (Received (1)) alıcı pin**, **TX (Transmit Transfer (0))** ise **gönderici pin** anlamındadır.

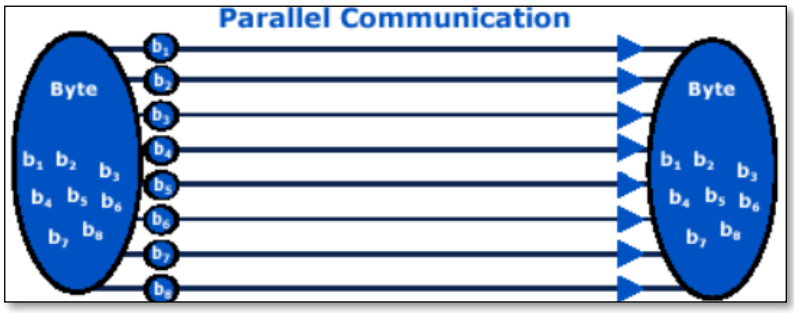
***Bu pinler aynı zamanda Arduino'nun bilgisayarla haberleşmesini sağlayan USB hattına da bağlıdır.***

*0 ve 1 numaralı pinler başka bir yere bağlı olduğunda, Arduino bilgisayarla haberleşme sağlayamamaktadır. Bu yüzden Arduino'ya kod atarken bu pinlerin bir yere bağlı olmamasına dikkat edilmelidir.*

Gönderilecek sinyaller paralel veya seri olarak taşınır.

1. **Paralel İletişim**

- Aktarılacak n bitlik kod olduğunu varsayalım, her bir bit ayrı bir iletim yolundan aktarılır.

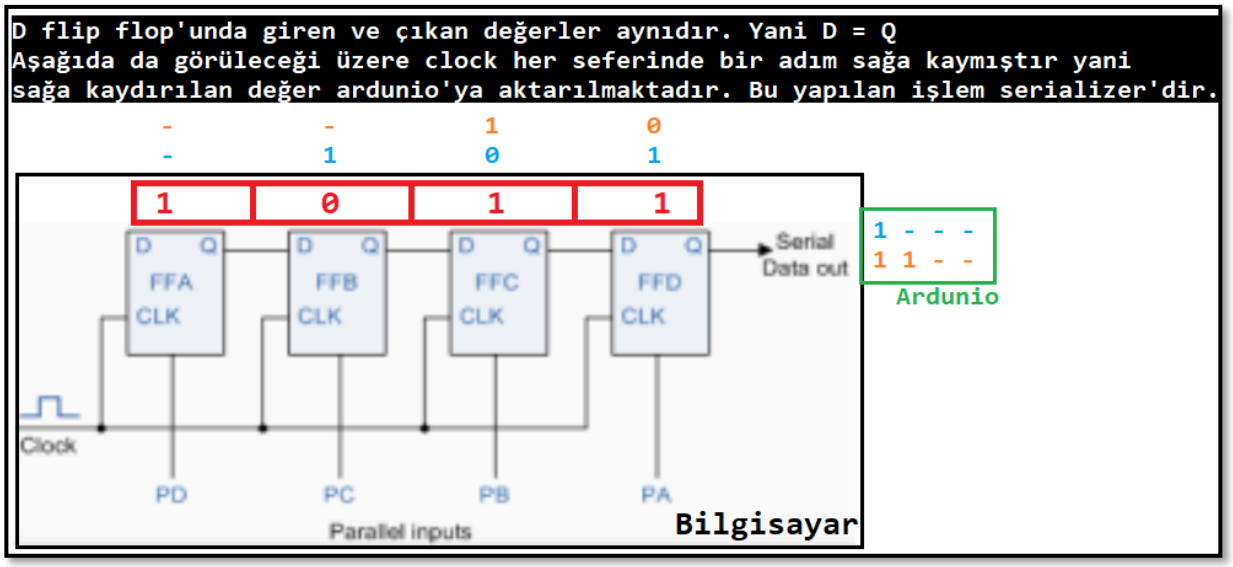



Paralel İletişimin Özellikleri:

- **Paralel iletişimin seri iletişimden hızlı olmasının nedeni**; seri ile gelen verinin paralele çevirme işleminin yapılmamasıdır. Bu çevrim işlemini yapan devrelere;
  - **Serializer:** Paralel taşınan veriyi seri hale getirir.
  - **Deserializer:** Seri taşınan veriyi paralel hale getirir.

  **Paralel iletişim serializer devresine ihtiyaç duymadığı** için hem hızlıdır hemde basit bir donanıma sahiptir.
- Kısa mesafeli iletişimler için uygundur.
- Paralel iletişimin, seri bağlantıya kıyasla daha yüksek bir kablo maliyeti vardır.
- Serializer ve Deserializer'ın tasarımında Flip-Flop kullanılmaktadır. Tasarımda flip-flop'un türü farketmemektedir.

- Aşağıdaki örnekte verici paralel devreye örnek gösterilmiştir.



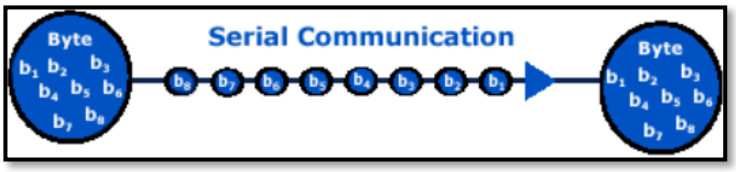

Paralel İletişimle İlgili Sorunlar (Dezavantajları)

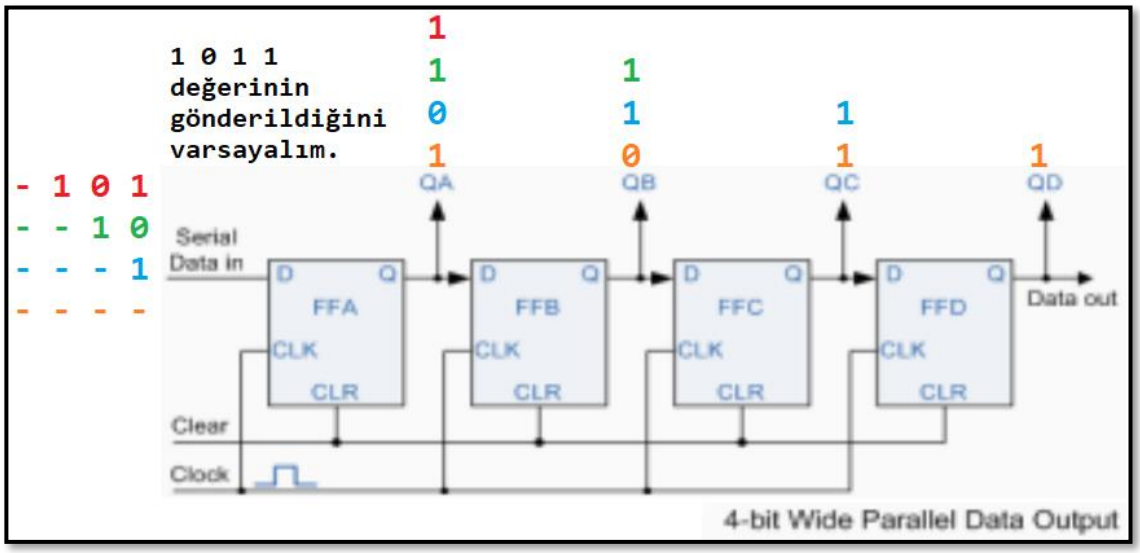
a. **Clock Skew (Saat Çarpıklığı)**
   Saat sinyali farklı bileşenlere farklı zamanlarda ulaşır. Bu da verilerin aynı anda gitmesini engeller. Küçük sapmalar oluşur. Clock süresi, uzun süren clock'a göre ayarlanır.

b. **Crosstalk (Hatların Karışması)**
   İletilen kablodaki sinyal uzun mesafelerde, diğer kablolardaki elektromanyetik sinyaller ile karışır. Bu da veri kaybına ya da veri değişimine yol açar.

## 2. Seri İletişim

- Gönderici ve alıcı arasındaki tek bir kanal üzerinden (tek kablo veyahut kablosuz ağ üzerinden) sırayla bit veri gönderme işlemi gerçekleşir.
- Doğru veri iletimi için, verici ve alıcı arasında bir senkronizasyon şekli olmalıdır.
- Seri iletişim kendi içerisinde asenkron ve senkron olmak üzere ikiye ayrılır.




### Seri İletişimin Özellikleri:

- Seri iletişimde bütün veri tek bir hat üzerinden gittiği için port sayısı azdır ve bu yüzden de kablo maliyeti azdır.
  Ayrıca daha az kablo bağlantısı gerektirdiği için daha az yer kaplar.
- Seri bağlantılar, daha yüksek bir veri hızına sahip olabilir.
- Uzun mesafeli iletişimler için uygundur.
- Seri iletişimde iki cihaz arasında dört kablo vardır:
  1. Transmit (TX)  2. Recevier (RX)  3. Güç (-)  4. Güç (+)

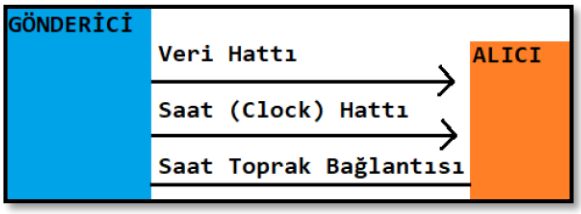
| | Paralel | Seri |
|---|---|---|
| **Crosstalk** | Kablo uzunluğunu sınırlar. | Daha az iletken kullanması nedeniyle bir sorun olmaz. |
| **Clock Skew (Saat Kayması)** | Veri aktarım oranını yavaşlatmak. | Asenkron iletişim için bir sorun değil. |
| **Kablo Uzunluğu** | Kısa | Uzun |
| **Maliyet** | Yüksek | Düşük |
| **Yapı** | Basit | Karmaşık (serializer/deserializer) |

- Hem verici hem de alıcı aynı baud hızını kullanır.
- Seri iletişim hızı iki şeye bağlıdır:
  - **Bit hızı:** Saniyede gönderilen bit sayısı (BPS).
  - **Baud hızı:** Saniyede gönderilen sembol sayısı. Sembol birden fazla bitten oluşabilir.

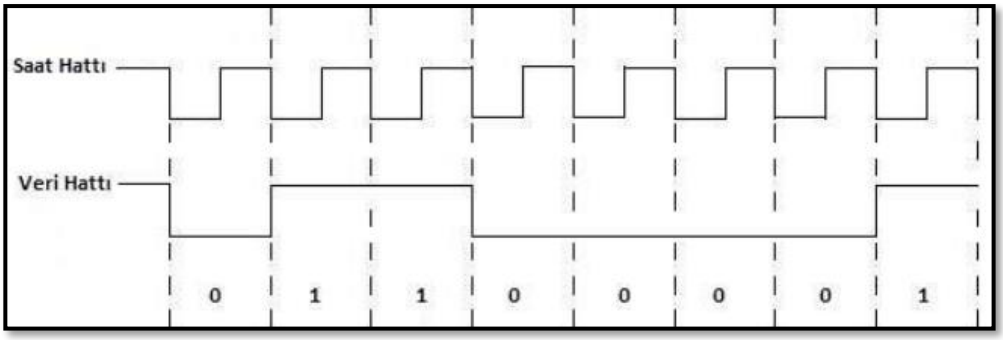
  Baud'daki sembolleri, ağ dersindeki paketlere benzetilebilir.
  Genelde bütün mikrodenetleyicilerde bit hızı ve baud hızı eşittir çünkü her clockta birden fazla veri gönderemiyoruz.

**a. Senkron Seri İletişim**

- Senkron iletişim, veri alma - gönderme işini eş zamanlı olarak yapmaktadır.
- Senkron seri iletişimde veri alışverişinin gerçekleştirildiği veri hattı, iki cihaz arasındaki senkronizasyonu sağlayan **clock (saat) hattı** ve **bu saat hattının toprak bağlantısı** bulunmaktadır.
- Veriler saat hattı tarafından gerçekleştirilen tetikleme yardımı ile iletilir. Bir cihazdan diğerine veri iletimi esnasında bir bitlik kısım gönderildiğinde saat hattı tetiklenir ve bu tetikleme ile alıcı cihaz tarafından bu bit alınır.



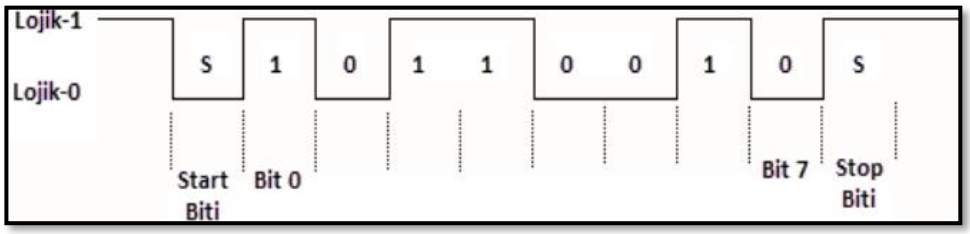

- Aşağıdaki şekilde saat sinyalinin tetiklemesi sonucu veri gönderimi görülmektedir. Noktalı yerlerde gösterilen değişimlerde veri iletimi gerçekleşir.




- Senkron iletimde, gönderilen verilere start ve stop bitleri konulmaz.
- Senktron iletişim hem paralel hem de seri iletişimde görülür.
- Veri kaybı olmaması için clock eşitlenmesi gerekir. Bu yüzden TX ve RX dışında bir de clock ayarı için bir kanal vardır. Bu da en yavaş clock'a göre ayarlanır.
- Senkron seri iletişimde veri iletim hızı yüksektir ancak maliyeti yüksek olduğundan kullanılmaz.

**b. Asenkron Seri İletişim**

- Asenkron seri iletişim, senkron seri iletişimden **farkı,** sadece veri hattı ile iletişimin gerçekleştiği, bir saat hattına gerek duyulmayan seri iletişim türüdür.
- Verinin başladığı ve bittiği noktaların bilinmesi için başlama **(start "0")** ve bitiş **(stop "1")** bitleri kullanılmaktadır. Ayrıca verinin düzgün iletilip iletilemediğinin kontrolü için **parity biti** kullanılır.
- Aşağıdaki şekilde 8 bitlik yani 1 byte uzunluğundaki verinin iletim şeması görülmektedir. Start ve stop biti ile toplam olarak 10 bit şeklinde gönderilir. Verinin her 8 bitlik kısmına bu işlem uygulanarak 10 bit olarak iletim gerçekleşir.




- Senkronizasyon, sabit bir Baud hızı kullanılarak ve başlangıç - bitiş bitleri kullanılarak yapılır.
- Her kelimede (WORD) senkronizasyon yapılır.

**NOT:** Senkron denildiği zaman mikrodenetleyici ile sensör,
Asenkron denildiği zaman mikrodenetleyici ile bilgisayar- mikrodenetleyici aklımıza gelmeli.

- Devrenin hızlı çalışması için kapılara ihtiyaç duyarız.
  Register'in, RAM'den hızlı olmasının sebebi kapılarla çalışmasıdır.
- **Projelerimizde** Arduino'ya komut yollamak veya sensörlerdeki değerleri görüntülemek için **seri haberleşmeyi kullanırız.**
  RF, Bluetooth ve USB üzerinden bilgisayara veri aktarmak için seri haberleşme protokolleri kullanılır.
- Arduino'nun USB kablosu üzerinden bilgisayara veri aktarmak için öncelikle haberleşme hızını (BaudRate) ayarlanmalı. Haberleşme hızı setup fonksiyonu içerisinde ayarlanmalıdır.

**X ve Y Paralel Verilerinin Seri İletime Çevrilmesi**

X ve Y verilerimiz paralel iletimle taşınıyor olsun. Bu verilerin seri iletilebilmesi için flip-flopla seri iletimle, iletilebilecek duruma çevrilmesi gerekir.

X ve Y ilk olarak **Serializer** ile serileştirilir. Bu ise şöyle yapılır:

1. X ve Y verileri bir register'in sağında ve solunda tutuluyor olsun.
2. Serializer devremizin içinde iki tane flip-flop olsun.
3. Sağ register'deki Y verisi sağdaki flip-flop'a sol taraftan girer.
4. İçerisinde çevrim yapılır ve flip-flop'un sağ tarafından çıkar.
5. Soldaki register'deki X verisi de ilk flip-flop'a sol taraftan girer.
6. İçerisinde çevrim yapılır ve flip-flop'un sağ tarafından çıkar.
7. Y verisi sağ tarafta olduğu için X verisi, seri iletişim yoluna aktarılamaz. X, önce Y verisinin karşı tarafa ulaşmasını beklemelidir.
8. Y verisi seri iletim yolundan karşı tarafa ulaşır.
9. Karşı tarafa ulaşan verinin de paralel iletişime uygun hale getirilebilmesi için Deserializer devresindeki flip-floplar ile işlem yapılması gerekir.
10. Y verisi, ilk flip-flop'a girer ve paralelize şekle getirilir.
11. Flip-flopun sağ tarafından çıkar.
12. Karşı tarafın ilgili register'inin sol yarısına Y değeri yazılır.
13. Artık X, serializer'in sol tarafından yola çıkarak iletime hazırdır.
14. X yola çıkar ve Deserializer'e gelir.
15. Bu arada X'in taşınma işlemi bir başka ikinci clock'ta gerçekleştiği için Y, sağ deserializer'in sağ flip flopunun sağına geçer. Y'nin işi bitmiştir.
16. Sol flip-flop'a giren X, paralelize edilir. Daha sonra ikinci flip-flop'a gelir.
17. Az önce registere yazılan Y verisi, o registerin sağ tarafına kaydırılır.
18. Gelen X verisi ise Y'nin az önceki yerine yani sola gelir.

`Serial.available()`, Arduino'nun serial buffer'ına (önbellek) o anda gelen verinin karakter sayısını yani uzunluğunu (ya da byte'ını) döndürür.
Seri portun üzerinde bir değer yoksa 0 dönecektir.
Herhangi bir argüman almaz.

```
void loop() {
  // Biz hiçbir şey yazmasak bile sonsuza kadar Merhaba yazar.
  while(Serial.available() >= 0){
    Serial.println("Merhaba");
  }
}
```

**********

`Serial.begin(9600)`, seri portla (bilgisayarla) iletişimin 9600 baud hızında olacağı belirtiliyor. Ayrıca seri iletişim için gerekli ayarlamalar yapılmaktadır.
`Serial.end()`, seri portu devredışı bırakarak RX ve TX pinlerini dijital giriş/çıkış olarak kullanılmasına izin verir. Bu fonksiyondan sonra tekrar seri portu etkinleştirmek için Serial.begin() fonksiyonu kullanılır.

**********

Seri porta Arduino aracılığıyla herhangi bir veri yazmak için `Serial.print("")` veya `Serial.println("")` fonksiyonlarını kullanıyoruz.

`Serial.println("")`, `Serial.print("")` komutundan farklı olarak, gönderilen yazıdan sonra satır başı yapılmasını sağlıyor.

```
Serial.print(78)            // 78
Serial.print(1.23456)       // 1.23
Serial.print('N')           // N
Serial.print("Hello world.") // Hello world.
Serial.print(78, BIN)       // 1001110
Serial.print(78, OCT)       // 116
Serial.print(78, DEC)       // 78
Serial.print(78, HEX)       // 4E
Serial.println(1.23456, 0)  // 1
Serial.println(1.23456, 2)  // 1.23
Serial.println(1.23456, 4)  // 1.2346
```

**********

`Serial.parseFloat()` ve `Serial.parseInt()`fonksiyonları seri porta gönderilen veri dizisi içerisindeki ondalıklı ve tam sayıları almamızı sağlar.

**********

`Serial.read()`, serial buffer'da sıradaki veriyi okur ve döndürür. Serial.read() ile **char** veri tipindeki verileri okuruz. Eğer biz **int** bir değer okuyacaksak okuyacağımız o veriyi 48'den çıkarmalıyız çünkü program char'dan okuduğu değeri ASCII tablosundan yararlanarak int'e çevriliyor.

```
void loop() {
  // Biz hiçbir şey yazmasak bile sonsuza kadar Merhaba yazar.
  while(Serial.available() >= 0){
    // 3 yazdigimizi varsayarsak: Degerimiz: 51
    // 5 yazdigimizi varsayarsak: Degerimiz: 53 yazacaktir.
    // Halbuki biz 3 ve 5 yazmasini istiyoruz, bu yüzden asagida yorum satirinda yer
alan kodu yazmaliyiz.
    int deger = Serial.read();
    delay(300);
    Serial.print("Degerimiz: ");
```

```
    Serial.println(deger);
    // Serial.println(deger - 48);
  }
}
```

////////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

```
char gelenDeger;
void setup() {
  Serial.begin(9600);
  while(Serial.available() <= 0){
    // Bir şey yapma.
  }
  gelenDeger = Serial.read(); // Char olarak okuyacak.
  Serial.println(gelenDeger);
  gelenDeger = gelenDeger - 48;
  for(int i = 0; i <= gelenDeger; i++){
    Serial.println(gelenDeger);
  }
}
// gelenDeger yerine 5 yazarsak öncelikle yazdığımız değer olan 5'i bize gösterir.
// Ardından alt alta 0, 1, 2, 3, 4, 5 rakamlarını yazar.
```

////////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

Eğerki gönderilen veriyi `Serial.parseInt()` ile okursak herhangi bir şey ekleyip çıkarmamıza gerek kalmadan klavyeden girilen veriyi direk öğrenebiliriz.

```
void loop() {
  // Biz hiçbir şey yazmasak bile sonsuza kadar Merhaba yazar.
  while(Serial.available() >= 0){
    // 3 yazdigimizi varsayarsak: Degerimiz: 3
    // 5 yazdigimizi varsayarsak: Degerimiz: 5 yazacaktir.
    int deger = Serial.parseInt();
    delay(300);
    Serial.print("Degerimiz: ");
   Serial.println(deger);
  }
```

////////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

Bilgisayardan gönderdiğimiz verinin Arduino tarafından nasıl okunacağını ve tekrar bilgisayara nasıl göndereceği aşağıda gösterilmiştir.

```
char data; // Bilgisayardan gelen veriyi saklar
void setup(){
  Serial.begin(9600); // Seri iletişim için ayarlar yapılıyor.
}
void loop(){
  // Bilgisayardan gelen veriler tekrar bilgisayara gönderiliyor.
  if(Serial.available() > 0){ // Bilgisayardan gelen veri varsa
    data = Serial.read();     // Gelen verinin 1 byte'ı okunup data'ya aktarılır
    Serial.print(data);       // data verisi bilgisayara gönderiliyor.
  }
}
```

Gelen veri read() metodu ile okunur ve data değişkenine aktarılır. read() metodu her seferinde sadece 1 byte veri okur ve okunan veri serial buffer'dan silinir.

**********
Serial.write(""), fonksiyonu print fonksiyonuna benzer bir işlev görür. Tek byte büyüklüğündeki değerleri yazdırmakta, String, char veya byte tipinde dizileri yazdırmakta kullanılır.
Alacağı ikinci bir parametre ile de yazdırılan String veya dizinin kaç elemanının yazdırılacağı ayarlanabilir.
Ayrıca write fonksiyonu geri dönüş değeri olarak üzerinde işlem yapılan byte sayısı döndürür.
Print; tipi farketmeksizin, parametre olarak aldığı veriyi byte dizisine dönüştürür ve öyle seri portuna yani buffer'a yazma işlemi yapar. Ancak write, direkt parametre olarak byte dizisi alır. Yazılacak byte dizisi belli ise write() kullanılmalıdır.

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.write(45); // 45 değeri gönder (byte)
  delay(2000);
  int gonderilenbytes = Serial.write(123); //metin gönder ve metnin uzunluğunu gönder
  delay(2000);
  Serial.println(gonderilenbytes);
}
```

```
//////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
// Eğer write içine birden çok değer göndereceksek bunları bir dizi
// içinde göndermeli ve write fonksiyonuna ikinci parametre olarak da
// dizi uzunluğunu geçmeliyiz.
int dizi[4] = {1,2,3,4};
Serial.write(dizi, 4); // Bunun sonucunda bize 4 değerini return eder

void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.write(45); // 1 byte kapladığı için 1 değeri döner
  int gonderilenbytes = Serial.write(123); //metin gönder ve metnin uzunluğunu dön.
  Serial.println(gonderilenbytes); // 1
}
```

**********
Setup fonksiyonunda kullanılan pinMode() metodu, iki parametre almaktadır. Birinci parametre hangi pin için işlem yapılacağını ifade ederken ikinci parametre ise belirtilen pinin giriş (INPUT veya INPU_PULLUP-READ) mi çıkış mı (OUTPUT-WRITE) olacağını belirtir.

**Düğme Kullanımı**

Düğme, arasında az bir boşluk bulunan iki tel gibi düşünülebilir. Kullanıcı düğmeye bastığında bu boşluk kapanır ve düğme iletken duruma geçer, üzerinden akım akar. Kullanıcı düğmedan elini çektiğinde devrenin eski konumuna dönmesi için, **PULL UP** ve **PULL DOWN** denilen direnç bağlantıları kullanılır. PULL UP ve PULL DOWN direnç ismi değil, dirençlerin bağlanma şeklidir. Genellikle 10K ohm direnç kullanılır.

**********
digitalWrite(), iki parametre almaktadır. Birinci parametre, hangi dijital pin üzerinde işlem yapılacağını belirtir. İkinci parametre ise birinci parametrede belirtilen pinin dijital **yüksek(HIGH-1)** mı ya da **düşük(LOW-0)** mu olacağını belirtir. Bir dijital pin yüksek yapıldığında, o pinden 5V gerilim ölçeriz. Dijital pin düşük yapıldığında ise o pinden 0V gerilim (GND) ölçeriz.

**pinMode ve digitalWrite komutlarında görüldüğü gibi 2 kelimeden oluşmaktadır. 2 kelimeden oluşan komutlarda ikinci kelimenin baş harfi büyük yazılır.**

```
// Seri portunden ac geldiginde ledi yaksin kapat geldiginde ise ledi kapatsin.
char gelenChar;
String gelenString;
void setup() {
  Serial.begin(9600);
  // 13 numarali pin cikis olarak ayarlandi.
  pinMode(13, OUTPUT);
}

void loop() {
  while(Serial.available() > 0){
    gelenChar = Serial.read();
    gelenString = gelenString + gelenChar;
  }
  if(gelenString == "ac"){
    // 13 numarali pin'e 1 degerini verdi.
    digitalWrite(13, HIGH);
    gelenString = "";
  }
  if(gelenString == "kapat"){
    digitalWrite(13, LOW);
    gelenString = "";
  }
}
```

**********

digitalRead(),belirli bir dijital pinin değerini HIGH veya LOW olarak okur. Parametre olarak okumak istediğimiz dijital pin numarasını gireriz.

```
// 2 no'lu dijital pinden buton durumu okunur.
digitalRead(2):
```

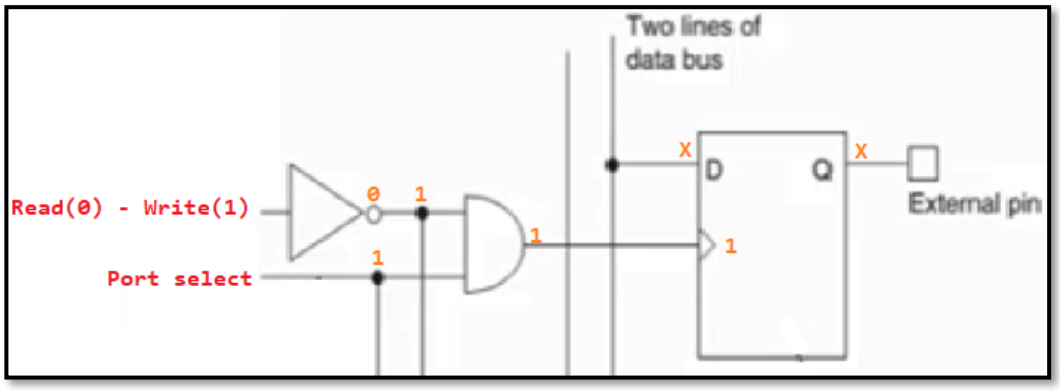
**********

random(max) şeklinde kullanırsak 0 ila max değeri arasında sayılar üretir.
random(min, max) şeklinde kullanırsak 0 ila max değeri arasında sayılar üretir

```
int sayi = random(20, 100);
```

sayi değeri 20 ila 100 arasındaki sayılardan birini alabilir.

## Arunio'da Okuma ve Yazma İşlemleri Nasıl Gerçekleşir




Kullanmadan önce I / O pininin yönünü belirlemelisiniz, bu TRIS, DDR registerının değerini değiştirerek yapılır.

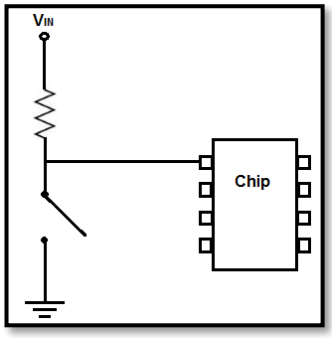
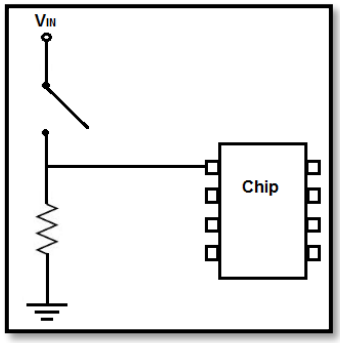
**TRIS Komutu:**

Portları giriş ya da çıkış olarak ayarlamak için kullanılır. Tris komutu hangi port için kullanılacaksa o portun harfi TRIS kelimesinden sonra getirilir. Örnek A portunu giriş ya da çıkış olarak kullanmak için TRISA, B portu için TRISB komutu kullanılır. Burada TRIS komutunun alabileceği iki değer vardır. Bunlar 0 ve 1'dir. TRIS komutu 0 ise port çıkış, 1 ise port giriş olarak kullanılır.

| | |
|---|---|
| TRIS**X** = **Değer**;<br>**X** = Mikrodenetleyicideki portun adı (A, B, C, D, E)<br>X en fazla kelime uzunluğunun değeri kadar olabilir.<br>**Değer** = 0 (Çıkış-Okuma) ya da 1 (Giriş-Yazma)<br>**Değer;** Binary, Hex, onluk sistemde olabilir.<br>Kaç port varsa o kadar TRIS vardır. | **Portun tüm pinleri için;**<br>TRISB = 0b00000000; Port B'nin **bütün pinlerini** binary olarak çıkış ayarla.<br>TRISB = 0x00; (hexadecimal olarak)<br>TRISB = 0; (decimal olarak)<br>**Portun belli bir pini için;**<br>TRISB.f3 = 1; Port B'nin 3. pinini giriş olarak ayarla. |

- Yazma işlemi, belirlenen PIN'e voltaj göndermektir aslında.
- Yazma yani devreden, dış bir sensöre değer yazma işlemi şöyle gerçekleşir:
  1. Yazma ve okuma devresi bulunmaktadır. Yukarıdaki şekilde sol tarafta bulunan iki girişten birincisi, işlemin okuma (0) mı yazma (1) mı olacağını belirler. Bu bit, kapılardan geçerek PIN'e ulaşır.
  2. Sol taraftaki ikinci giriş ise görüldüğü üzere Port Select devresidir yani **pinMode()**. Bu devre yazma işlemi hangi PORT'da yapılacak ise o PORT'a 1, geri kalan bütün PORT'lara 0 değerini verir.
  3. Yazılacak olan veri, giderken register'deki değer de alınarak flip-flop'a gider. Flip-flop'ta yazılacak pin seçilir. O pin'e 1, kalan pin'lere 0 verilir. Daha sonra veri ilgili pine yazılır.
  4. Yazma modunda data bus'dan karşı tarafa değer gidiyor.

- Okuma işlemi, belirlenen PIN üzerinde bulunan durumu (voltaj seviyesini) okumak demektir.
- Okuma işlemi şöyle gerçekleşir:
  1. Yazma ve okuma devresi bulunmaktadır. Yukarıdaki şekilde sol tarafta bulunan iki girişten birincisi, işlemin okuma (0) mı yazma (1) mı olacağını belirler. Bu bit, kapılardan geçerek PIN'e ulaşır.
  2. Sol taraftaki ikinci giriş ise görüldüğü üzere Port Select devresidir yani **pinMode()**. Bu devre okuma işlemi hangi PORT'da yapılacak ise o PORT'a 1, geri kalan bütün PORT'lara 0 değerini verir. Ancak bu sefer register'deki değer okunup PIN'e yazılmayacak, PIN'deki değer okunup register'e yazılacak.
  3. Okuma modunda data bus'a değer gider.

| **Pull Up:** | **Pull Down:** |
|---|---|
| Butona basıldıktan sonra +5 Volt, daha az direnç bulunan yolu tercih edeceği için toprağa gidecektir ve entegre üzerine 0 volt seviyesinde bir gerilim düşecektir. | Düğmeye basıldığında gerilim kaynağıyla PIC girişi kısa devre olur.<br>Buton ile +5 Volt verdiğimizi varsayarsak lojik sistemin diğer ucunda yüksek değerde bir pull down direnci bulunduğu için dışarıdan verilen +5 Volt lojik sistemimize uygulanacaktır. |
|  |  |

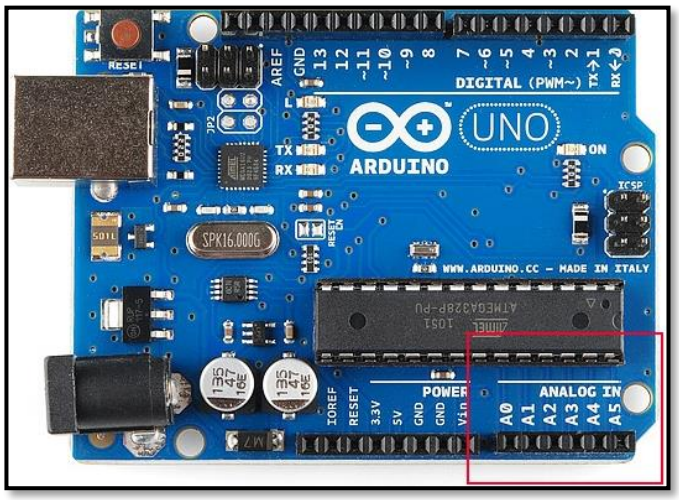
**SORU-1:** Her anahtara basıldığında led durumunu okuma – aktif hale getirme ???

```
int dugmeDurumu = 0;
int ledDurumu = 0;
void setup() {
  pinMode(13, OUTPUT);
  pinMode(5, INPUT);
}

void loop() {
  // 5. pinin yani dugmenin durumunu surekli olarak aktariyoruz yani aciksa 1
kapaliysa 0
  dugmeDurumu = digitalRead(5);
  // Dugme aciksa 13. ledi aktif hale yani HIGH hale getir.
  if(dugmeDurumu == HIGH){
    if(ledDurumu == LOW){
      // HIGH yerine 1 yazsakta olur.
      digitalWrite(13, HIGH);
      ledDurumu = 1;
    }
  }
  else{
    digitalWrite(13, LOW);
    ledDurumu = 0;
  }
  // Motorun patlamamasi icin
  while(dugmeDurumu == HIGH){
    dugmeDurumu = digitalRead(5);
  }
}
```

## ADC (Analog Digital Converter)

- ADC, analog sinyallerin dijital sinyallere çevriminin gerçekleştirilmesidir.
- **"Ardunio UNO"** denetleyicisinde **"6"** adet **analog giriş** bulunur. Bunlar **"A"** portundadır. Eğer bu **pinleri aktif etmezseniz,** bu portları **normal "Dijital" giriş - çıkış birimi olarak kullanabilirsiniz**.




- "ADC" pinleri **"10"** bit değerindedir. Yani, **"$2^{10}$ = 1024"** 'ten yola çıkarak, voltaj değişimine göre **"0"** ile **"1023"** arası bize değer göndermektedir.
  **%V / 1024 =** 0,0024 volt ile her bir binary sayının temsil edebileceği gerilimi görmüş oluruz.
  **Gerilim = (( ADC Gerilimi * ADC_Sonucu) / 1023)**
  Bizim mikrodenetleyicimiz 8 bit ama ADC bize 10 bit sonuç değeri döndürebilir. Bunda herhangi bir sakınca yok çünkü iki tane register'i yan yana gelebilir (16 bit).

MC (V) / MC (A. Has. 2^has+1) = 0.00488V (4.9 mV)
5V / 1024 = 0.00488V (4.9 mV)
Gerilim = ((ADC Gerilimi * ADC_Sonucu) / 1023.0)

ADC Gerilimi = 5V (Arduino UNO)
ADC_Sonucu = Porttan gelen değer

ADC Özellikleri

1. **Analog girişte bit sayısı ne kadar fazla olursa o kadar hassas veri okunuşu gerçekleşir.**
2. **Pulse sayısının az olması yani az sayıda parçaya bölünmesi hassas veri okumamızı sağlar.**
3. **Pin sayısının fazla olması hassas veri okumamızı sağlar.**

- `analogRead()` metodu belirtilen pin'den analog voltajı okuyup binary olarak temsil eder. Parametre olarak pin numarasını alır. Dönüş değeri yani karşımıza çıkacak olan sonuç değeri 0-1023 arasındadır.

```
int analogPin = 3; // potansiyometre 3 nolu analog pine bağlanmıştır
int okunanDeger= 0;
void setup(){
  Serial.begin(9600); // Seri port bağlantı ayarı yapılıyor.
}
void loop(){
  okunanDeger = analogRead(analogPin); // analog değer pinden okunuyor.
  Serial.println(okunanDeger); // okunanDeger PC'ye gönderiliyor.
}
```

## 1. Analog Giriş Pini

Bir Arduino kartının analog sinyalleri işleyebilmesi için analog girişler kullanılır. Analog giriş sayısı, aynı anda kaç farklı kaynaktan analog veri alınabilecğeini belirler. Analog sinyaller, mikrodenetleyici tarafından işlenebilmeleri için dijital veriye dönüştürülürler.
Bu dönüştürme işlemi Arduino Uno kartında 10 bitlik bir dijital veriye dönüştürülme şeklinde gerçekleştirilir. 0-5V arasında analog sinyal üreten bir sensör 10 bitlik ADC tarafından dijitale dönüştürüldüğünde**, 0-1023 ((2^10)-1=1023)** arasında bir sayıyı temsil ederken 12 Bitlik ADC tarafından bir dijital veriye dönüştürüldüğünde, 0-4095 ((2^12)-1=4095) arasında bir sayı ile ifade edilebilir. Dolayısıyla 12 bitlik dijital verinin analog veriyi daha hassas tanımlayabildiğini görebiliriz. Analog giriş pinleri, ihtiyaç olduğunda dijital pin olarak da kullanılabilmektedir.

Analog pinler de diğer dijital pinler gibi en fazla 5 volt gerilime dayanabilmektedirler. Bu pine 5 voltun üzerinde bir gerilim uygulanırsa, Ardunio bozulabilir. Kısacası ardunio'nun ölçebileceği en yüksek gerilim 5 volttur.

## 2. Analog Çıkış Pini

Analog çıkış pini ise, dijital bir verinin analog veriye dönüştürülmesi görevini yerine getiren Dijital Analog dönüştürücünün çıkış pinidir.

- `analogWrite()`, pin'e analog değer yazmak için kullanılır. Analog değerden kastımız, ayarlanabilir miktarda çıkışa 0-5 V arasında değerler vermektir.
  Diğer adı PWM (pulse with modulation) olan bu sinyalin amacı 5V'luk DC gerilimin, çıkışa belirli sürede verilmesidir. Bu sayede DC motorun hız kontrolü, led'in ışık yoğunluğu, ya da hassas kontrol gerektiren herhangi bir elektronik elemanın kontrolü sağlanabilir.

**Kullanımı:**

```
analogWrite(PIN, Value)
```

PIN: Kartınızda belirtilen PWM çıkış pin'lerinden herhangi biri.
Value: 0-255 arasında bir değer.

`analogWrite()`, komutunda 255 değeri maksimum çıkış voltajını yani 5V'u temsil ediyor. 0 ile 255 arası tüm değerler 0 - 5V arası voltaj değerlerine denk düşüyor. Örneğin `analogWrite(9, 80)` komutu, 9 numaralı pinden 5V x (80/255) = 1,57V geriliminde çıkış almamızı sağlıyor.

- `map()` metodu paramete olarak verilen bir değişkenin, değerinin sınır aralıklar içinde olmasına göre dışarı bizim belirleyeceğimiz aralıklarda bir değer döndürebilir.

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

**Parametreler**
value: Ölçümlenecek değişken,
fromLow: Değer aralığının alt sınırı,
fromHigh: Değer aralığının üst sınırı,
toLow: Hedef aralığının alt sınırı,
toHigh: Hedef aralığının üst sınırı,

```
// A0 pininden gelen volt degerini olcen kodu yaziniz.
void setup() {
  Serial.begin(9600);
}

void loop() {
 float gelenDeger = analogRead(0A); // 0-1023 arasindaki deger aktarilir.
  delay(500);
  Serial.print("Gelen Deger: ");
  Serial.println(gelenDeger);

  float gerilim = (gelenDeger * 5) / 1023;
  delay(500);
  Serial.print("Volt Degeri: ");
  Serial.println(gerilim);
  // Sicaklik degerini hesaplama
  float sicaklik = gelenDeger * 0.4883848;  // Onemli olan ilk iki basamak yani 5'e
yakin bir deger olmasi.
  Serial.print("Sicaklik Degeri: ");
  Serial.println(sicaklik);
}
```

**Sensör**

- ***Sensörler,*** çevredeki ışık, nem ve basınç gibi değerleri sinyallere dönüştürüp bize değerlerini veren cihazlardır.
- **Sensör Nasıl Seçilir**

  Algılanacak nesnenin seçilmesi lazım.
  Çevreye uyumlu olması lazım.
  Bağlantı şemasına bakılması gerekir.

Sensörlerin aralığı önemlidir. Örneğin bir sıcaklık sensörü hangi aralıktaki sıcaklıklara duyarlı olacak?
Bir sensörde en az 3 pin (bacak) olması gerekir. TX, + ve – pinleri.
TX ile bir bacaktan verileri alıp mikrodenetleyiciye taşırken "+" ve "-" ile de elektrik alır.

- **Pasif Sensörler (Passive Sensors)**

  Sinyal üretebilmek için dışarıdan harici hiçbir güç kaynağına ihtiyaç duymayan fiziksel ya da kimyasal değerleri istenilen çıkış değişkenine dönüştürebilen sensörlerdir. Bu pasif sensör çeşitlerine en basit örnek ise buton ve anahtardır. Bunlardan farklı olarak potansiyometre, limit anahtarları, ISI sensörleri (PTC ve NTC), basınç sensörleri, LDR, fototransistörler, fotodiyotlar ve mikrofonlar örnek olarak söylenebilir. Bu sensörlerin çalışması için harici hiçbir enerjiye ihtiyaç yoktur. Bu sensörler sadece giriş değişkenlerini ölçerek tepki verirler.
- **Aktif Sensörler (Active Sensors)**

  Sinyal üretebilmesi için dışarıdan harici bir güç kaynağına ihtiyaç duyan sensörlere **aktif sensör** denilir. Bu sensörlerin en önemli özelliklerinden biri düşük sinyalli ölçmelerde kullanılmasıdır. Bundan dolayı oldukça hassas ölçüm yapabilirler. Aktif sensörler, ürettiği sinyal türüne göre; Analog veya Dijital sinyal çıkışı vermektedirler. Dijital olarak 0 ya da 1 çıkışını vermektedirler. Çıkış sinyali analog olan sensörler ise gerilimsel yada akımsal çıkış verebilirler. Gerilim sinyali olarak genellikle 0-5V arasında bir gerilim vermektedirler. Akım sinyali olarak ise genellikle 4-20mA arası bir çıkış vermektedirler.

**Yüksek Seviyede Yazılan Programın İşlemci Tarafından İşlenmesi İçin Hangi Adımlar Gerçekleşmelidir? (Program Nasıl Çalışır?)**

Öncelikle mikroişlemcinin hangi mimariye sahip olduğunu bilmemiz gerekir.

1- ASM diline dönüştürülür. (Birinci sebep; tek komutta tek işlem yapıldığı için ikinci sebep; her komutta bir iş.)
2- Lincar (Bağlama) işlemi yapılır. (Eğer main'in üzerinde kütüphane varsa Lincar işlemi gerçekleşir aksi takdirde gerçekleşmez.)
3- .obj dosyası oluşur. (Memory organizasyonu ve işlemci uyumlu komutları barındırır.)

int a = 10; RAM'de gerçekleşir.
int b = 20; RAM'de gerçekleşir.
int c = a+b;
int a = c+b;
a, b ve c değerleri bizim anlamamız için vardır işlemci gözünden bakacak olursa bunlar adres anlamındadır.
Mikroişlemciye ilk önce c = a+b; komutu gitti.
Mikroişlemci ile memory veyahut RAM (emin değilim) arasında 4 adet yol vardır:
**veri yolu, veri adresi, komut yolu ve komut adresi.**

4- Komut getirme adımları. (Fetch – Decode – Execute)

**Bir program ASM diline ne zaman dönüştürülür?**

1- Çalıştırıldığı zaman
2- Kaydedildiği zaman

İşletim sistemi kaydettiğimiz programı ya ilerde çalıştırırsak diye ASM diline dönüştürür.

**Kaydetmenin İçinde Ne Var?**

1- Derlenir.
2- ASM diline dönüştürülür.
3- Makine diline dönüştürülür.
4- Harddisk'e kaydedilir.

**Çalıştırmanın İçinde Ne Var?**

1- Derlenir.
2- ASM diline dönüştürülür.
3- Makine diline dönüştürülür.
4- Harddisk'e kaydedilir.
5- Memory'ye yüklenir ardından işlemci tarafından işlenir.

Assembly yürütülebilir bir program oluşturmaz. Yürütülebilir program için aşağıdakiler gereklidir:

**Program Çalıştırma Adımları**

**1-** Assembly diline dönüştürülür. (Kod Binary formatına (makine diline) dönüştürülür.)
**2-** Linking gerçekleştirilir. (Kütüphaneye gidip komutları getirir.)
**3-** Loading gerçekleştirilir. (Program, RAMe taşınır.)
**4-** Execute gerçekleştirilir.

Uygulamaya her tıkladığımzda 'loading' olur çünkü uygulamanın komutları belleğe yüklenir.
Execution aşaması da gerçekleşip uygulama çalıştırılır.

**SINAVDA PROGRAM ÇALIŞTIRMA ADIMLARI SORULURSA EXTRA OLARAK KOMUT ÇALIŞTIRMA ADIMLARINIDA YAZMAYI UNUTMA!**

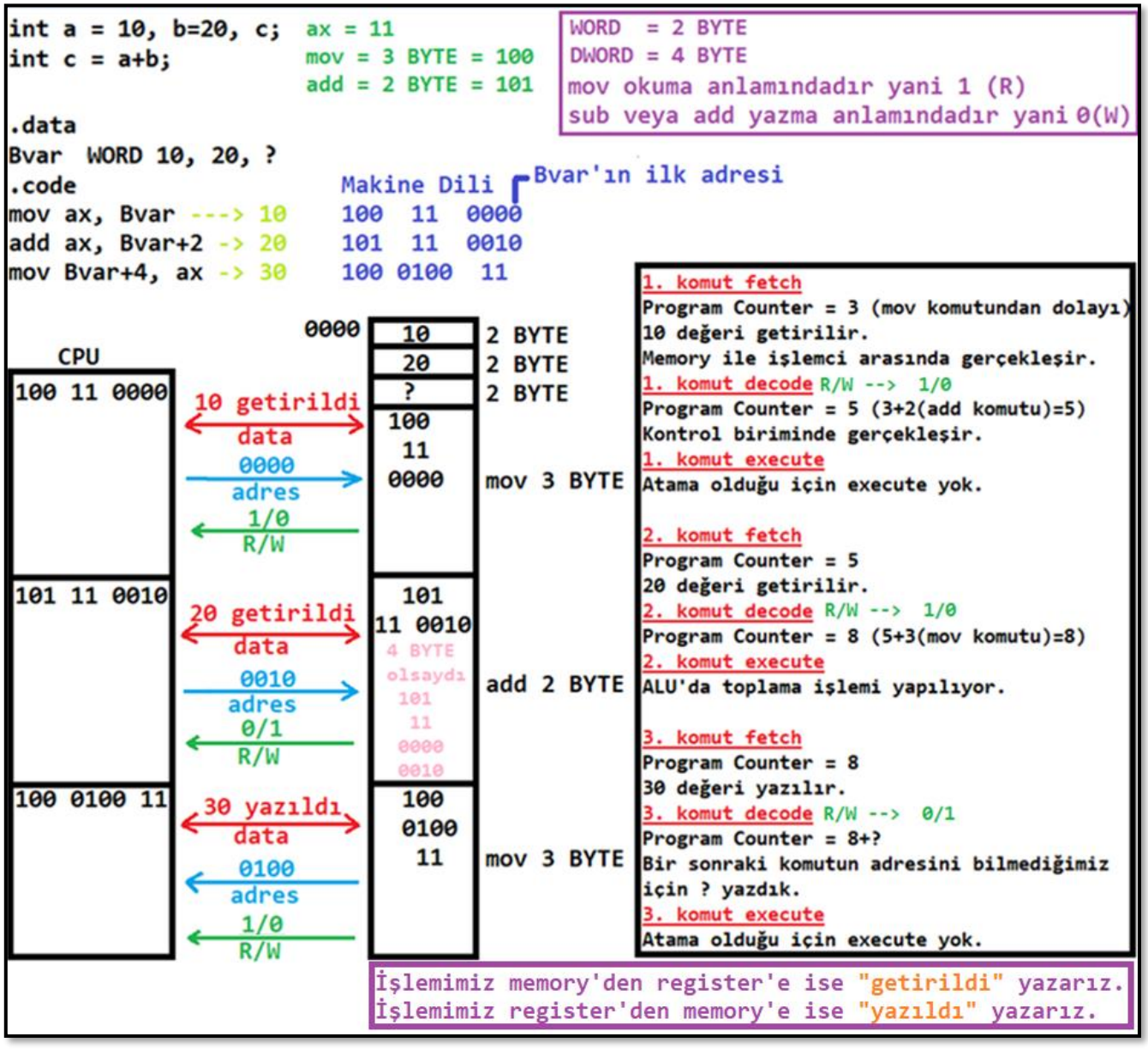
**Komut Çalıştırma Adımları**

**1-** Komut getirilir. **FETCH (İşlemci ile hafıza arasında ya da 2 tane memory arasında gerçekleşir.) (RAM'den kaydedicilere gider.)**
**2-** Kod çözülerek anlamlı hale getirilir. **DECODE (Kontrol birimi tarafından gerçekleştirilir.)**
**3-** Komut neyse o çalıştırılır. **EXECUTE (Execute, ALU veya çıkış portlarında gerçekleştirilir.)**

Komutlar önbelleğe kaydedilmez çünkü önbellekte sadece değişkenler tutulur.
Fetch değişkenlerde daha çok yapılır.

**ASM Çevirme Adımları**

1- Değişken taraması yapılır, türleri ile birlikte (string, int) getirilir.
2- Değişkenler sınıflarına göre kümelenir.
3- Memory organizasyonu yapıldıktan sonra makine diline dönüştürülür.
4- Komut çalıştırma adımları (fetch-decode-execute) ve en az bir komut bir defa fetch yapılması lazım.
5- Program Counter her seferinde bir sonrakinin adresini içerecek şekilde otomatik olarak arttırılır.



**Timer (Zamanlayıcı) Nedir?**

- Timer **belirli zaman aralıklarında** yapmak istediğimiz olayların gerçekleşmesinde bize yardımcı olan yapıdır. Örneğin belirli zaman aralıklarında herhangi bir çıkışa bağlı led'i yakmak için **timer** kullanırız ancak timer kullanmadığınızı varsayarsak **delay** fonksiyonuyla o an işlemciyi bloke edebiliriz.

**Timer Nasıl Çalışır?**

- Timerlar **sayaç yazmacı (counter register)** değişkenini periyodik olarak arttırırlar. Bu yazmaç belirli bir değere kadar ulaşabilir. Bu değeri belirleyen faktör ise kullandığımız timer'ın kaç bit olduğudur. Genellikle 8 bit veya 16 bit olarak karşımıza çıkarlar. Bu yazmaç maksimum değerine ulaştığı zaman (8 bit ise bu değer 255 çünkü 0'dan başlıyor) sıfıra resetlenir ve taşmanın olduğuna dair bir flag bit'i set edilir. Bu işlemi Interrupt Service Routine (ISR) gerçekleştirerek bizim kesme olduğundan haberdar olmamızı sağlar. Bu flag bit'i için gereken tüm kontrolleri ISR sağlamaktadır, peki timer counter değişkenini periyodik olarak artırır dedik. Bu periyotu neye göre referans alıp arttırır? Timer çalışması için bir clock sinyaline ihtiyaç duyar. Bu sinyal her oluştuğunda bu değişkeni bir arttırır. Bu sinyal harici olabileceği gibi dahili olarakta çoğu işlemcide mevcuttur.
- Zamanlayıcıları overflow bitiyle kontrol edebiliriz.
- Herhangi bir mikrodenetleyicinin zamanlayıcı konusuna baktığımızda aşağıdakileri bilmemiz gerekir:
    1. **Timer bit sayısı**
    2. **Timer sayısı**
    3. **Timer mode (mode)sayısı**
    4. **Timer prescaler çeşitliliği**

**Interrtup (Kesme) Nedir?**

- Interrupt fonksiyonu, ana programın yapmakta olduğu işle ilgili bir görevi varsa, ana programın yaptığı işi askıya alır ve kendi kodlarını çalıştırır.
  Diyelim ki Arduino'nun Interrupt özelliği bulunan bir düğmesine basıldığında bilgisayara veri yollamak istiyoruz. Fakat Arduino başka işlemler yaparken kullanıcı düğmeye basabilir. Böyle bir durumda Arduino düğmeye basıldığını anlayamayacaktır. Böyle bir hatanın önüne geçilmesi için Interrupt kullanılmalıdır. Düğmeye atanacak bir Interrupt, Arduino başka bir işlem yapmakta olsa bile, düğmeye basıldığı gibi Arduino'ya haber verecektir. Arduino düğmeye basıldığında yapılması gereken işlemleri yaptıktan sonra, kaldığı yerden diğer işlemlere otomatik olarak geri dönecektir.
- Arduino'da farklı görevlerde kullanılmak üzere çeşitli Interrupt'lar (kesmeler) bulunur. **Zaman kesmesi (timer interrupt)** ve **dış kesmeler (external interrupt)** en yaygın olarak kullanılan Arduino kesmeleridir.

**1- Zaman Kesmesi (Timer Interrupt)**

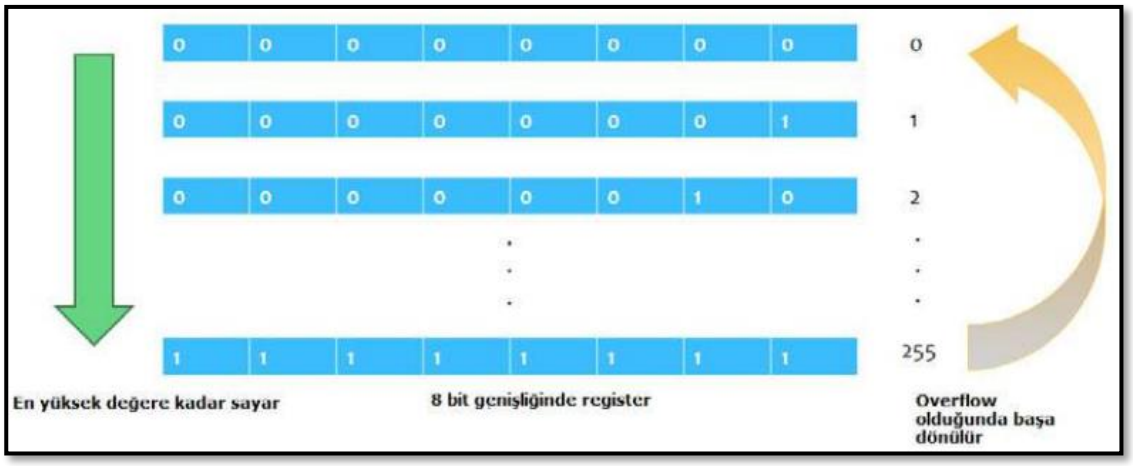
- Zaman kesmesi (timer interrupt), belirli süre aralıklarında belirli görevlerin yapılabilmesi için kullanılır. Örneğin bir LED'in saniyede bir yakıp söndürülmesi gerekmektedir. Kullanılan kesme her saniyede bir Arduino'ya haber vererek, LED'in yakılıp söndürülmesini sağlayacaktır. Zaman kesmelerinin ayarlanabilmesi için bazı program değişkenlerinin ayarlanması gerekir. Bunlardan en önemlisi zaman sayıcısıdır. Bu değişkende tutulan değer, her saat darbesinde bir artar. Taşma değerine ulaştığında sıfırlanır ve Arduino'ya sürenin dolduğunu, kesmenin yapılması gerektiğini bildirir. Bu değişkenin başlangıç değeri ayarlanarak, kesmeler arasındaki geçecek zaman ayarlanabilir.
- Farz edelim ki, 8 bitlik zaman sayıcısının ilk değeri 50 olarak belirlendi. Her adımda bu değer bir artırılacaktır. Zaman sayıcısının değeri 255 olduktan bir adım sonra sıfırlanacaktır. Bu noktada zaman kesmesi gerçekleşecektir. Kesme gerçekleştiğinde Arduino otomatik olarak "`ISR(TIMERx_COMPA_vect)`" fonksiyonunu çalıştıracaktır. Bu fonksiyonun içerisine her kesmede yapılması istenilen görevler yazılır.
- Zaman kesmesinin ne kadar sürede bir gerçekleşeceği, her adımın ne kadar sürede gerçekleşeceğiyle de alakalıdır. Bu süre Arduino'nun kristalinin 1, 1/8, 1/64, 1/256 ve 1/1024 oranında olması sağlanabilir. Zaman kesmesinin oluşma süresi aşağıdaki formül ile hesaplanabilir.

**sayacDegeri = [(16000000 / prescaler ) * istenen interrupt frekansı] – 1**

**Millis fonksiyonu,** Arduino'nun kaç milisaniye süresince çalıştığı döndürür. Değerler milisaniye cinsinden tutulduğu için geri döndürülen değer int veya float'ın tutabileceği kapasitenin çok üstünde olabilir. Bu yüzden bu fonksiyonla birlikte unsigned long türündeki değişkenler kullanılır.

## Timer Interrupt (Zaman Kesmesi) Nasıl Kullanılır?

- Timer kesmeleri belirli süre ile programlanabilir istenilen zamanda tetiklenebilir. Genel olarak Timer'ı alarmlı saate benzetebiliriz. Belirli bir süre için saat kurulur vakti geldiğinde saat normal çalışmasını devam ettirirken aynı zamanda alarm çalışır.
- Timer'lar, **counter register** denen bir **register'e** sahiptir. Bu register'ın görevi timer'ın sayacı olarak çalışmaktır. Timer her adımda maksimum değerine ulaşıncaya kadar bu sayacı arttırır. Sayaç maksimum değeri aşıldığında değeri sıfırlanır. Timer'ın **overflow (taşma)** bayrak biti set edilir. Bu anda interrupt'ların tetiklendiği anda çalıştırdığı fonksiyon olan **ISR (Interrupt Service Rutin)** çalışır.




## Timer Çeşitleri

- Arduino UNO kartında Atmega 328 mikrodenetleyicisi kullanılmıştır. Atmega 328'in 3 timer'ı vardır. Timer 0, Timer 1 ve Timer 2.

1. **Timer0:** 8 bitlik bir timer'dır. 8 bitlik olduğu için 0-255 arasında değer verilerek kullanılabilir. `delay()` ve `millis()` gibi fonksiyonlar bu timer'ı kullanır.
2. **Timer 1:** 16 bitlik timer'dır. Timer sayacı 0 ila 65535 aralığında değerler alabilir. Arduino Servo kütüphanesi bu timer'ı kullanır.
3. **Timer 2:** Bu timer'da Timer0 gibi 8 bitliktir. Arduino `tone()` fonksiyonunda kullanılır.

## Tımer Ayarları ve Çalıştırma

- Arduino UNO kartında ki bu üç timer'ı kullanabilmek için, öncelikle ayarları yapmalı ondan sonra çalıştırmalı. Timer register'larından **TCCRxA** ve **TCCRxB register'ları** timer'ı ayarlamak için kullanılır. **TCCR** Timer/Counter Control Registeri (zamanlayıcının sayaç kontrol register'ı) anlamındadır. Her register 8 bit değer tutar ve her bit bir konfigürasyon değerini saklar.

Tablo 7.2 - TCCR1A Timer/Counter Control Register A ve TCCR1B Timer/Counter Control Register B nin her bitinin görevleri

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| (0x90) | COM1A1 | COM1A0 | COM1B1 | COM1B0 | - | - | WGM11 | WGM10 |
| Read/ Write | R/W | R/W | R/W | R/W | R | R | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| (0x81) | ICNC1 | ICES1 | - | WGM13 | WGM12 | CS12 | CS11 | CS10 |
| Read/ Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Örneğin Timer1'i kullanabilmek için **TCCR1B'nin** son üç biti ayarlanmalı. Bunlar **CS12**, **CS11**, **CS10'dur**. Timer'ın clock ayarları bu bitler ile ayarlanır. Bu bitlerin değişik varyasyonlar da ayarlanmasıyla timer farklı hızlarda çalıştırılabilir. Aşağıdaki şekilde gösterilmiştir.

| CS12 | CS11 | CS10 | AÇIKLAMA |
|---|---|---|---|
| 0 | 0 | 0 | Clock kaynağı yok/ Timer sayacı çalışmıyor |
| 0 | 0 | 1 | $clk_{I/O}$/1 (No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (From prescaler) |
| 1 | 0 | 0 | $clk_{I/O}$/256 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (From prescaler) |
| 1 | 1 | 0 | Harici clock kaynağı T1 pini.Clock düşen kenar |
| 1 | 1 | 1 | Harici clock kaynağı T1 pini Clock Yükselen kenar |

- Timer'lar 3 modda kullanılabilirler. **Normal Mod**, **CTC modu** ve **PWM** modudur. Timer interrupt'larında kullandığımız mod CTC modudur. CTC; timer, istenen süreye eriştiğinde ISR fonksiyonunu tetikleyip çalışmasını sağlayan moddur. CTC (Clear Timer On Compare) modunda timer'ın iki değeri vardır. Birisi **Set point (Eşik Noktası)** diğeri **Process Value (İşlenen Değer)'**dir. Process value sürekli olarak Set point ile karşılaştırılır. Process value, set point'e ulaşır ya da aşarsa process value resetlenir. Timer karşılaştırma sonunda sıfırlanır. CTC modunu kullanabilmek için **TCCRxA** ve **TCCRxB** register'ları, **OCRxA** ve **OCRxB** register'ları, **TIFR** ve **TIMSK** register'ları kullanılır.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | COM1A1 | COM1A0 | COM1B1 | COM1B0 | FOC1A | FOC1B | WGM11 | WGM10 | TCCR1A |
| Read/Write | R/W | R/W | R/W | R/W | W | W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Şekil 7.5 - TCCR1A register'ı

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 | TCCR1B |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Şekil 7.6 - TCCR1B register'ı

## 2- Dış Kesmeler (External Interrupt)

- Dış kesmeler, Arduino'nun özel pinlerinde gerçekleşen voltaj değişimlerini takip eden kesmelerdir. Örneğin Arduino'nun dış kesme pinine bağlanmış bir düğmeye basıldığında, dış kesme Arduino'ya otomatik olarak haber verir. Bu kesme sayesinde Arduino sürekli olarak düğmeye basılıp basılmadığını kontrol etmek zorunda kalmaz, bu sırada başka işlemleri yerine getirebilir. Aşağıdaki tabloda Arduino türlerine göre dış kesme özelliğine sahip pinler gösterilmiştir.

| KART | INT0 | INT1 | INT2 | IN3 | INT4 | IN5 |
|---|---|---|---|---|---|---|
| UNO | Pin2 | Pin3 | - | - | - | - |
| MEGA2560 | Pin2 | Pin3 | Pin21 | Pin20 | Pin19 | Pin18 |

Tablo 7.1 - Bazı Arduino kartlarının Interrupt pinleri

- Kullanılacak dış kesme pini, setup fonksiyonu içerisinde `attachInterrupt(0, fonksiyon, mod)` metotu ile aktifleştirilmeli. Burada 0 ile belirtilen *ilk parametre*, **INT0** olarak belirtilen interrupt'dır. Ardunio UNO için bu 2 numaralı pindir. *İkinci parametre* kesme durumunda çalışacak fonksiyonu belirler, *üçüncü parametre* ise interrupt'ın nasıl tetikleneceğini belirtir yani aşağıdaki 4 moddan birisi seçilebilir.

- Aşağıda "attachInterrupt()" fonksiyonu için tanımlanabilecek dış kesmelerin türleri belirtilmiştir.

    1. **LOW:** Interrupt pini LOW seviyesinde olursa, yani pindeki voltaj 0 ise kesme oluşur.
    2. **CHANGE:** Interrupt pinine uygulanan gerilim değiştiğinde, LOW iken HIGH olduğunda veya HIGH iken LOW olduğunda kesme oluşur.
    3. **RISING:** Yükselen kenarlarda kesme gerçekleşir. Yani pindeki voltaj değeri 0'dan (LOW) 5 Volta (HIGH) çıktığında kesme gerçekleşir
    4. **FALLING:** Düşen kenarlarda kesme gerçekleşir. Yani pindeki voltaj değeri 5'ten 0'a düştüğünde kesme gerçekleşir.

  Bu modlardan herhangi birisi seçilerek istenen donanım interrupt'ının tetiklenmesi sağlanır. Interrupt tetiklendiğinde ilgili fonksiyon çağrılır.

- Interrupt'ları pasifleştirmek için `noInterrups()` metodu kullanılırken, pasif olan interrupt'ları tekrar aktifleştirmek için ise `interrupts()` metodu kullanılır. Interrupt'ın kapatılması için ise `detachInterrupt(interrupt)` metodu kullanılır.

- Dış kesme fonksiyonları **void()** türünden olmak zorundadır yani geriye bir değer döndürmez.
  Dış kesme fonksiyonları içinde `delay()` ve `Serial.available()` fonksiyonları çalışmaz.
  Aynı anda birden fazla Interrupt kullanmak istediğinizde bu Interrupt'ların hepsi tetiklense dahi, aynı anda sadece bir fonksiyon çalıştırılacağından, birinin çalışması bittikten sonra sıradaki fonksiyon çalışmaya başlayacaktır.

- **Zaman Paylaşımı (Time-Sharing):** işlemci zamanı, kullanıcıların çoklu erişimlerine imkan sağlayabilmek için çok küçük zaman dilimlerine bölünmekte, her zaman diliminde bir tek kullanıcıya hizmet verilmektedir. Mikroişlemcilerde timesharing yoktur. Bu yüzden mikroişlemcilerde zamanlayıcılar kullanılır.
- Mikrodenetleyicileri birbirinden ayıran özellik, karşılaştırma özelliğinin olup olmamasıdır.

PROJEDE ANALOG, ZAMANLAYICI, EEPROM, KÜTÜPHANE, HABERLEŞME HERHANGİ 3 TANESİ KULLANILACAK.

- Timer'i register kontrol eder. Timer; shift left, shift right yaparak zamanı artırabilir.
- Timer sayısı ve timer'in bit sayısı, mikrodenetleyicileri birbirinden ayıran özelliklerdendir.
- Timer istediğimiz değere vardığında yani kesme olduğunda **Interrupt Flag 1** olur.
- **Global Interrupt Enable:** Tüm kesmeleri çalıştırır. Sadece timer diye bir kesme yoktur. Birçok kesme vardır. Bir harici cihazı takmak dahi kesmedir.
- Interrupt enable 1 olduğunda Interrupt Service Routine fonksiyonu (tipi void'dir) çağrılır. Interrupt enable'nin 1 olması için herhangi bir kesme olayının olması gerekir. Kesmenin türü önemli değildir.
- Bazı mikrodenetleyicilerde EEPROM (Program memory) ile birleşik ama bazı mikrodenetleyicilerde de ayrı olmak üzere Interrupt Service Routine için bir depolama alanı vardır.
- Dört depolama türü var demiştik. Ancak Interrupt Service Routine'e özel olarak EEPROM bölgesinin(flash memory) içinde bu fonksiyon için bir alan ayrılmıştır.
- delay() kullandığımızda arduino belirtilen süre boyunca yeni bir işi yerine getiremez. Timer interrupt, delay() fonksiyonundan etkilenmez. Delay komutu ile işlemci atıl duruma geçse de timer çalışmaya devam eder. Yani timer paralel olarak çalışır. Kod çalışmaya devam ederken timer çalışmaya devam eder.
- Program ALU'da yapılırken Timer kendisi artırma işlemini yapar(timer için register bu işlemi yapar (shift left, shift right)) yani programdan bağımsızdır. Timer'in zamanı işlemcinin clock'una bağlıdır.
- Timer registerine mesela 100 değerini verdik. Her clock'ta bir artırılır. Her değer arttığında registerdeki değere eşit mi diye kontrol eder. Eğer eşitse timer interrupt devreye girer. Interrupt Flag da 1 olur. Register'deki mesela 100 sayısı aslında bizim süremizdir.
- **Clock dispathcer:** Normalde her clock'ta timer 1 artırılır. Ancak dispatcher ile şunu diyebiliriz: 4 clock'ta 1 artır.

**EEPROM**

- EEPROM, byte bazında erişimin yapıldığı bir hafıza tipidir. Bu diske kendimiz manuel bir şekilde direkt olarak istediğimiz adreslere yazma ve o adreslerden okuma yapabiliriz. Güç kesilmesi durumunda bile bu veriler silinip gitmemektedir.

- EEPROM'un belli bir veri tutma kapasitesi bulunmaktadır. 0-1023 arasında bir adres bulunmaktadır ve bu adresler 1 byte'lık bir veri tutmaktadır. Aşağıdaki fonksiyoları kullanabilmek için `#include <EEPROM.h>` şeklinde kütüphanemizi eklememiz lazım.

- İlk olarak adreslemeyi bir hatırlayalım. Mesela 1 KB boyutundaki bir disk kaç byte'dır? 2^10 byte'dır. Yani bu diskin adresleme kapasitesi 2^10'dur yani 1024'tür. En son ise 1023 adresine kadar adresleme yapar çünkü ilk adres 0'dan başlamıştır.
  EEPROM'da belli bir adresteki veriyi okuyabiliriz. 0-1023 arasında bir değer okuyabiliriz.
  `int gelenVeri=EEPROM.read(512);` şuan 512. adreste hangi veri varsa onu getirir.

- `EEPROM.write(512, 255);` 512. adrese 255 sayısını yazacak.
  Bir byte 8 bittir. Yani maximum 255 sayısına kadar tutabilir bu yüzden bir adrese 255'ten daha yüksek bir sayı girildiğinde hata vermese de yanlış sonuçlar verecektir.

- Read ve Write metotları, **sadece bir byte boyutunda okuma ve yazma yapar**. Peki elimizde bir byte'dan daha büyük veriler varsa ne yapacağız? Bunun için **GET** ve **PUT** metotları mevcut.
  Örneğin float'ın boyutu 4 byte olduğundan float'ın sonuna kadar gidecek ve 4 byte'lık okuma yapacaktır.

  Ayrıca Read'den farklı olarak, okunan verinin, fonksiyona verilen ikinci parametreye kendiliğinden atanması. Yani Read'deki gibi `gelenVeri=EEPROM.read(512)` ile veriyi okuyup manuel olarak bir değişkene atamak yerine GET, bunu bizim için istediğimiz bir değişkene kendisi atıyor.
  ```
  float x;
  EEPROM.get(512, x);
  ```
  512 adresindeki float değişken, x değişkenine atandı

  PUT'da GET'e benzer şekilde çalışır. Yazılmak istenen verinin boyutu 1 byte'ı geçiyorsa geçen byte'lar sonraki adreselere sırasıyla yazılır.
  ```
  float x = 42.42;
  EEPROM.put(512, x);
  ```
  x değişkenindeki değeri 512 adresine yazar. Float 4 byte olduğu için kalan 3 byte'ı da sırasıyla 513,514,515 adreslerine yazar. Sırasıyla 513, 514, 515 adreslerine yazdığından da emin değilim ama öyle tahmin ediyorum

- Arduino kartında, hemen yanında çizgi olan pinler, analog olarak da kullanılabilecek olan pinlerdir. Bazı pinlerin hem dijital hem analog olarak kullanılabileceğini söylemiştik.

## Arduino'da Kütüphane Oluşturma

➢ Kütüphane oluştururken farklı iki uzantılı dosya kullanacağız portayarı.cpp ve portayari.h gibi.
**portayari.h** : bu dosya header dosyamız yani yazılacak fonksiyonların deklarasyonunu yapıyoruz.
**portayari.cpp** dosyası ile header dosyası içerisinde deklare ettiğimiz metotların yapacağı işlemleri yazıyoruz.

**Öncelikle portayari.h adında header dosyamızı oluşturalım; dosya uzantısı .h**

1. Header dosyası #ifndef dosya_adı_H ifadesi ile başlar ve #endif ifadesi ile biter kullanılan tüm komutlar, kütüphaneler ve sınıflar belirtilen iki ifadenin arasında yazılır.
2. Kullanılan kütüphane varsa #include <kütüphane_adı> ifadesi ile eklıyoruz.
3. Sınıf oluşturulur ve sınıfın içerisinde fonksiyonların sadece adları yazılır.

**portayari.h dosyasının içeriği**

portayari.h | portayari.cpp

```
#ifndef portayari_H
#define portayari_H

#include <Arduino.h>

class portayari {
public:
    portayari(int pin1);
    void port_high();
    void port_low();
    void port_sure(int sure);
private:
    int _pin1;
};

#endif
```

4. 1. ve 2. Satırdaki kodlarla header dosyamızın adını yazıyoruz ve sonuna resimdeki gibi _H ekliyoruz.
5. 3. satırda ise arduinonun özelliklerini kullanabilmek için header dosyasını dahil ediyoruz.
6. 4. satırda ise yine header dosyamıza ait sınıfımızı oluşturuyoruz ve bu sınıf içinde değişkenlerin ve fonksiyonların isimlerini tanımlıyoruz.
7. Sınıf içerisinde bulunan fonksiyonları portayari.cpp dosyası içerisinde açıklıyoruz.

**Fonksiyon dosyası oluşturma dosya uzantısı .cpp**

- Kullanılan header dosyalarını #include "dosyaadı.h" ifadesi ile tanımlıyoruz.
- İstenen fonksiyon veya fonksiyonlar yazılıyor (fonksiyon isimleri header dosyasındaki fonksiyon isimleri ile aynı olmalıdır).

**portayari.cpp dosyasını oluşturma.**

portayari.h | portayari.cpp

```
#include "portayari.h"
#include "Arduino.h"


portayari::portayari(int pin1){
    pinMode(pin1, OUTPUT);
    _pin1 = pin1;
}


void portayari::port_high(){
    digitalWrite(_pin1, HIGH);
}

void portayari::port_low(){
    digitalWrite(_pin1, LOW);
}

void portayari::port_sure(int sure){
    digitalWrite(_pin1, HIGH);
    delay(sure);
    digitalWrite(_pin1, LOW);
    delay(sure);
}
```