C ile AVR Programlama ve Gömülü Sistemlere Giriş

Gökhan Dökmetaş

Atmel AVR Mikrodenetleyiciler ile Gömülü Sistem Tasarımı Eğitimi

İçindekiler

Hazırlanacaktır..
Gönüllüler Yazar ile İletişime Geçebilir.

Söz Başı

C ile AVR programlama ve Gömülü Sistemlere giriş kitabı www.lojikprob.com adlı kişisel internet sayfamda yazdığım "C ile AVR Programlama" yazı dizisinin derlenmesiyle meydana gelmiştir. Bu yazı dizisinin tek bir yerde toplanmasıyla daha okunabilir ve kullanışlı bir içerik meydana gelmesi hedeflenmiştir. En önemli hedeflerimden biri ise bundan sonra yazacağım kitapları ücretsiz e-kitap olarak yayınlamaktı. Bu dosyanın hiçbir zaman bir kitap kalitesinde olduğunu iddia etmesem de kısıtlı zamanım olduğu için elimden gelenin en fazlası budur. AVR mikrodenetleyicilerin ülkemizde yeterli üne kavuşamamasının başlıca nedeninin hakkında yeterli Türkçe kaynak olmamasından dolayı olduğunu düşünüyorum. Bu alandaki Türkçe kaynak eksikliğini gidermeye yönelik olan bu çalışmanın size faydalı olacağını umut ediyoruz.

Benimle iletişime geçmek için aşağıdaki e-posta adresini kullanabilirsiniz,

gokhandokmetas0@gmail.com

AVR Nedir?

AVR, Atmel firması tarafından tasarlanıp 1996'dan beri piyasaya arz edilen mikrodenetleyici ailesinin adıdır. Bu mikrodenetleyiciler modifiye edilmiş Harvard mimarisi üzerine RISC komut kümesiyle tasarlanmıştır. Mikrodenetleyiciler 8-bit olup istisna olarak bir dönem 32-bit modelleri üretilmiştir. Gömülü sistemlerde ve özellikle hobi devrelerinde en çok kullanılan mikrodenetleyicilerden olup Arduino geliştirme platformunun da temelini oluşturmuştur.

Ülkemizde AVR mikrodenetleyiciler, PIC mikrodenetleyiciler ile beraber en yaygın kullanılan iki mikrodenetleyici ailesinden biridir. Dünyada ise özellikle batıda PIC mikrodenetleyicilerden daha yaygın kullanılmaktadır. Bunun sebebi ucuzluğu ve ücretsiz geliştirme ortamı sağlamasından dolayıdır. Birim maliyeti ucuz olduğundan seri üretimi yapılan cihazların içerisinde AVR mikrodenetleyicileri görmek mümkündür.

Resim: https://protostack.com.au/wp-content/uploads/IC-ATMEGA328-PU.jpg

AVR Mikrodenetleyici Aileleri

Avr mikrodenetleyici aileleri aşağıda sayacağımızdan daha fazla olsa da işimize yarayacak ve günümüzde üretimi hala devam eden olanlardan bahsetmekle yetindik. Bu mikrodenetleyici aileleri arasında temelde bir fark olmayıp aynı geliştirme ortamı üzerinden programlama yapılsa da özellik ve kullanım alanı bakımından fark bulunur.

tinyAVR

Entegrelerin üzerinde ATtiny olarak kodlanan bu ailedeki mikrodenetleyicilerin özellikleri şöyledir, – 0.5 – 32 KB Program Hafızası

-6-32 pin arası entegre kılıfı

-Sınırlı arayüz tabanı

megaAVR

Entegrelerin üzerinde ATmega olarak kodlanan bu ailedeki mikrodenetleyicilerin özellikleri şöyledir, – 4-256KB Program Hafızası

-28-100 pin arası entegre kılıfı

-Genişletilmiş Komut kümesi

– Geniş arayüz tabanı

XMEGA

Entegrelerin üzerinde ATxmega olarak kodlanan bu ailedeki mikrodenetleyicilerin özellikleri şöyledir,

-16-384KB Program Hafızası

-44-64-100 pin entegre kılıfı

-Artırılmış performans özellikleri

-Artırılmış arayüz tabanı

Bundan başka 32-bit AVR ve FPSLIC yani FPGA ile AVR'yi birleştiren bir yapı olsa da bunlar artık bulunmamaktadır.

PIC mi AVR mi ?

Bu sorunun kesin cevabı olmamakla beraber benim şahsi görüşüme göre AVR mikrodenetleyiciler çok büyük sebeplerden dolayı olmasa da PIC mikrodenetleyicilerden üstündür. Sebeplerini madde madde yazarsak,

1. PIC mikrodenetleyiciler bir komutu işlemek için 4 çevirime ihtiyaç duyar. Yani 16 MHz'de çalışan bir mikrodenetleyici aslında 16/4 yani 4MHz'de çalışmaktadır. AVR Mikrodenetleyiciler ise bir komutu işlemek için çoğunlukla bir çevrime ihtiyaç duyar. O yüzden AVR mikrodenetleyiciler daha hızlıdır.
2. PIC mikrodenetleyiciler aynı özellikteki AVR mikrodenetleyicilere göre daha pahalıdır. AVR mikrodenetleyiciler ucuzdur.
3. AVR mikrodenetleyicilere başlamak için oldukça ucuza Arduino kartları bulabiliriz. Arduino kendi başına bir platform olsa da donanımsal olarak bir geliştirme kartı olarak kullanılabilir.
4. AVR'nin Avrfreaks gibi uzun yıllardan beri canlılığını koruyan toplululuğu ve internette kütüphane, kaynak kod gibi alanlarda büyük bir birikimi vardır.
5. AVR'nin Visual Studio tabanlı tam C dili desteği veren AVR Studio geliştirme ortamı vardır. PIC'de ise uzun yıllar resmi olarak sadece assembly derleyicisi kullanıldı ve üçüncü parti birbirinden farklı C derleyicileri günümüzde hala kullanılmaktadır.

Çevre Birimleri

Mikrodenetleyiciyi mikroişlemciden ayıran en büyük fark içerisinde gömülü halde bulunan çevre birimleridir (ing. peripheral). Program hafızası, RAM, EEPROM, Zamanlayıcılar, Kesmeler, ADC gibi temel çevre birimlerinin çoğu hemen her AVR mikrodenetleyicide eksiksiz olarak bulunmaktadır. AVR mikrodenetleyicilerin her birinde ayrı bir özellik olan bir modeli yerine az model olup çoğu özelliğin toplandığı mikrodenetleyiciler olması bir mikrodenetleyiciyle hemen hemen tüm işleri yapmamıza imkan sağlar. O yüzden birkaç AVR mikrodenetleyiciyi öğrenmek bize yeterli gelecektir.

Kaynaklar :

AVR microcontrollers, Wikipedia, bağlantı: https://en.wikipedia.org/wiki/AVR_microcontrollers, Erişim Tarihi : 14.08.2018 PIC vs. AVR smackdown, ladyada.net, bağlantı: http://www.ladyada.net/resources/picvsavr.html, Erişim Tarihi: 14.08.2018

Uygun AVR Seçimi

Öğrenmek için seçeceğimiz mikrodenetleyici aynı zamanda gelecekteki projelerimizde kullanacağımız mikrodenetleyicidir. İçini dışını iyi bildiğimiz bir parçayı öncelikli olarak seçmek yeni parçayı öğrenme zahmetinden bizi kurtaracaktır. O yüzden öğrenmeye başlarken uygun mikrodenetleyiciyi seçmek çok önemlidir. Atmel AVR mikrodenetleyicilerde yanlış mikrodenetleyiciyi seçmek pek mümkün olmasa da PIC mikrodenetleyicilerden hepimizin bir yerden duyduğu 16F84 entegresinde ADC bile yoktur. Buna rağmen 16F84 üzerine pek çok ders yazılmış hatta kitap bile basılmıştır. Doğru olan ortanın biraz üstü seviyede bir entegre kullanmaktır.

AVR mikrodenetleyicilerde PIC mikrodenetleyicilerde olduğu gibi yüzlerce kod addan oluşan farklı entegreler yoktur. Çeşit az olup özellikler dağınık halde değil toplu haldedir. O yüzden uygun entegreyi seçmemiz oldukça kolay olmaktadır. Bizim derslerde kullanmak ve ileride proje yapmak için tavsiye ettiğimiz entegreler %99'a yakın işimizi görecek seviyede olacaktır.

Uygun AVR seçimi üst seviye denecek bir entegre seçiminin yanında yaygın olarak kullanılan entegrelerden birini seçmekle olur. AVR kodları birbirine benzese de özellikle ayak sayısı ve bazı özelliklerin kullanılması bakımından her proje için yazılmış kod her entegrede çalışmayabilir. Bunun için çoğu projeyi kodda ufak düzeltmelerle çalıştıracak kapasitede bir entegrenin seçilmesi gerekir.

Entegrelerin kılıfları aşırı derecede önem arz etmektedir. DIP (Dual Inline Package) kılıfta olmayan SMD entegreler yeni başlayanların pek sıcak bakmadığı, pahalı ekipmanlara ihtiyaç duyan, prototiplemeyi zorlaştıran yapıdadır. O yüzden yeni başlayanlar DIP kılıftaki entegreleri kullanmak zorundadır.

Seçilecek AVR 8-bit mikrodenetleyici ailesinin özelliklerinin çoğunu barındırmak zorundadır. Böylelikle mikrodenetleyici ile yapabileceklerimizin alanı genişlerken diğer entegrelere ait özellikleri de öğrenmiş oluruz. Bu özellikler özetle ADC, PWM, Zamanlayıcı, Kesmeler, USART, SPI, I2C gibi iletişim ve çevre birimleridir.

Şimdi derslerde ve projelerde kullanmak üzere seçtiğimiz üç entegreden bahsedelim.

Atmega328P

Bu entegreyi seçmemizin en büyük nedeni Arduino Uno kartında kullanılmasıdır. Pek çok Arduino kodu ve projesi Atmega328P üzerine yazıldığından internette oldukça geniş kaynak bulmamız mümkündür. Ayrıca ATmega mikrodenetleyiciler arasında özellik ve kullanışlılık bakımından en üst seviyede olan entegrelerden biridir. Mikrodenetleyicinin özellikleri şu şekildedir,

Ayak Adedi: 32

Program Hafızası : 32KB

SRAM: 2KB EEPROM: 1KB

ADC (bit) : 10

ADC (ch) : 6

PWM : 6 8-bit zamanlayıcı: 2

16-bit zamanlayıcı: 1

Atmega32A

40-pin DIP kılıfta olan Atmega32A fazladan ayak sayısı gereken durumlarda kullanılmaya uygundur. Ayrıca uzun zamandan beri hobicilerin tercih ettiği bir entegre olup hakkında pek çok kaynak kod ve proje vardır.

Ayak Adedi: 40

Program Hafızası : 32KB

SRAM: 2KB EEPROM: 1KB

ADC (bit) : 10

ADC (ch) : 8

PWM: 4 8-bit zamanlayıcı: 2

16-bit zamanlayıcı: 1

Atmega32u4

Arduino Leonardo'da da kullanılan bu entegre pek çok ATmega ailesinin ferdinde bulunmayan USB özelliği olması sebebiyle bazı projelerde kullanmak zorunda olacağımız bir entegre olacaktır. Bu entegrenin en büyük eksi yanı DIP kılıfının olmayışıdır. O yüzden SMD lehimleme ekipmanlarına ihtiyacımız olacaktır.

Ayak Adedi: 44

Program Hafızası : 32KB

SRAM: 2.5KB EEPROM: 1KB

ADC (bit) : 10

ADC (ch) : 12

PWM:4

8-bit zamanlayıcı: 2

16-bit zamanlayıcı: 1

Kataloglar

Mikrodenetleyici seçiminde Microchip tarafından yayınlanan resmi kataloglardan yararlanabilirsiniz.

8-bit AVR® Microcontrollers Peripheral Integration Quick Reference Guide
http://ww1.microchip.com/downloads/en/DeviceDoc/30010135D.pdf

8-bit PIC® and AVR® Microcontrollers
http://ww1.microchip.com/downloads/en/DeviceDoc/30009630M.pdf

Atmel Product Selection Guide
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-45154-Product-Selection-Guide_Brochure.pdf

Kaynaklar

ATmega16U4/32U4 Datasheet – Microchip Technology, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf, Erişim Tarihi : 14.08.2018

ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 14.08.2018

Atmel ATmega32A – Microchip Technology, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8155-8-bit-Microcontroller-AVR-ATmega32A_Datasheet.pdf, Erişim Tarihi: 14.08.2018

Donanım Seçimi

Bir mikrodenetleyici platformuna giriş için gerekli donanım ve yazılım elemanlarını temin etmek gerekir. Günümüzde yazılım İnternet sayesinde erişilebilir olsa da donanım yönünde son yıllara kadar yurt içi pazara bağlı konumdaydık. Üreticiden ya da yabancı tedarikçilerden satın almak da ödeme yöntemlerinin kısıtlanması ve döviz kurlarından dolayı oldukça maliyetli ve zor hale gelmiştir. Buna çözüm getiren tek yer Aliexpress gibi

çin menşeili sitelerdir. Bir platforma ait ürünleri Aliexpress'de bulabilmemiz o platformun tedarik sıkıntısını neredeyse ortadan kaldırır durumdadır. Bu başlıkta da birinci şart olarak donanımın Aliexpress'de ya da Türkiye pazarında bulunabilir olmasına göre konuya dahil edeceğiz.

Deneme Kartları ve Geliştirme Kartları

Mikrodenetleyici eğitim kartlarını iki ana sınıfa ayırmak istersek bunları deneme kartları ve geliştirme kartları olarak ikiye ayırırız. Deneme kartları eski usul olup üzerinde tuş takımı, LCD ekran, LED, buzzer gibi birçok elektronik elemanı üzerindeki mikrodenetleyiciye bağlı halde tek kart olarak sunan ve kullanıcıyı devre kurmak yerine sadece program yazmaya sevk eden kartlardır. Bu kartların olumsuz özelliği yenilenebilir olmaması, prototipleme yapılamaması, sadece tek bir yönde kullanılabilmesidir. Üstelik üzerinde bulundurduğu devre elemanları ve işçilikten dolayı geliştirme kartına göre oldukça pahalıdır. Üzerinde programlama devresini bulundurması ayrı bir programcı alma ihtiyacını ortadan kaldırsa da mikroişlemcilerin Ön yükleyici (bootloader) özelliği sayesinde bu avantaj ortadan kalkmaktadır.

Ön yükleyici (bootloader) özelliği mikrodenetleyiciyi devreden sökmeden ve seri iletişim yoluyla programın güncellenebilmesine olanak sağlayan bir donanım özelliğidir. Mikrodenetleyicide ana program önceden olağan programlama ile program hafızasına yüklenir. Bu ana program ön yükleyici görevde olup hafızada küçük bir yer tutar. Seri iletişimde aldığı verileri program hafızasının geri kalan kısmına yükleyerek çalışacak programın (uygulamanın) seri iletişim üzerinden yüklenebilmesine olanak sağlar.

Deneme Kartı Resim:
https://www.pantechsolutions.net/media/catalog/product/cache/1/image/600×600/9df78eab33525d08d6e5fb8d27136e95/p/i/pic-development-board-08.jpg

Geliştirme kartları ise temel olarak bir mikrodenetleyiciyi çalıştırmak için asgari gereklilikte parçayı üzerinde bulundurup mikrodenetleyicinin ayaklarına kolay erişim sağlatan kartlardır. Bazı geliştirme kartları en sık kullanılan ve ihtiyaç duyulan elemanları üzerinde barındırabilir. Programlama devresi, USB yuvası, LED bunlardan başlıcalarıdır. Bazı geliştirme kartlarının kendilerine ait derleyicisi hatta programlama dili bile bulunur. Bunlara

geliştirme platformu denir. Arduino, BASIC Stamp gibi geliştirme platformları hobiciler için hem yazılım hem donanım olarak oldukça kolaylaştırılmış tam kapsamlı platformlardır. AVR programlamayı istediğimizden dolayı geliştirme platformlara mümkün olduğunca yazılım yönünden uzak kalmayı tercih edeceğiz. Yalnız AVR ile Arduino artık oldukça birbirine yaklaştığı için kısmen Arduino kütüphanelerinden ve kodlarından yardım alabiliriz. Çünkü Arduino'nun temeli AVR donanımı ve yazılımıdır.

Bir geliştirme kartı olarak Arduino UNO

Resim: https://store-cdn.arduino.cc/usa/catalog/product/cache/1/image/520×330/604a3538c15e081937dbfbd20aa60aad/a/0/a000066_featured_4.jpg

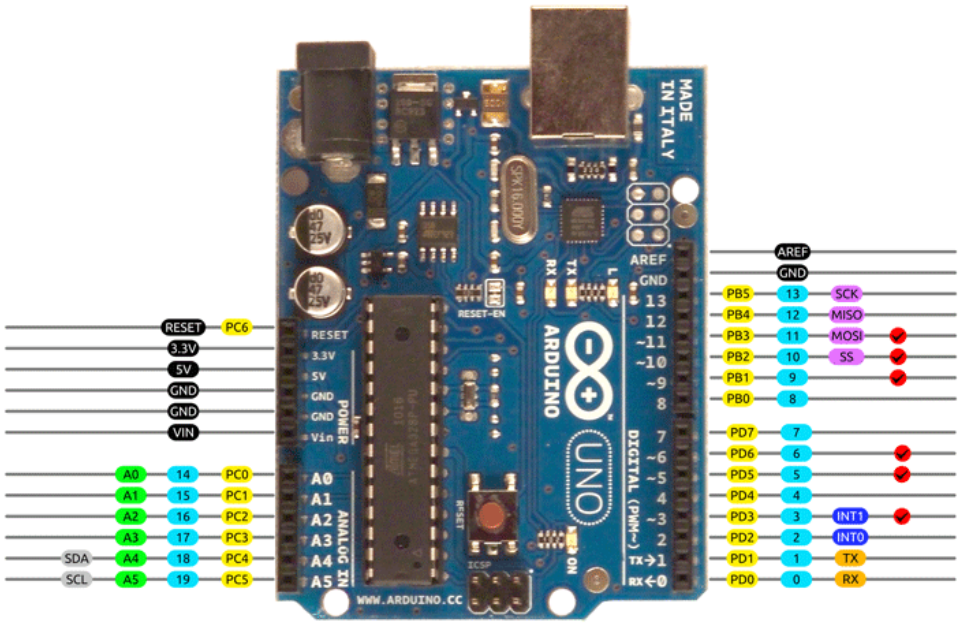
AVR için üreticinin sağladığı resmi geliştirme platformları mevcuttur. Fakat bunların maliyeti alternatiflerine göre oldukça yüksek olup fiyatına göre özellikleri yeterli değildir. O yüzden basit bir prototipleme kartı ya da geliştirme kartı ile başlamak en akıllıca çözümdür. AVR için üretilen üçüncü parti geliştirme ve deneme kartlarını Çin sitelerinde bulmak mümkündür. Bu kartların Arduino'ya göre ciddi eksiklikleri vardır. Pek çoğuna USB üzerinden program atılamamakta ve harici bir programlayıcı istemektedir. Harici programlayıcının yanı sıra ekstradan bir güç adaptörüne ihtiyaç duymaktadır. Arduino hem gücünü USB üzerinden almakta hem de aynı yoldan program atılabilmektedir.

Çok uygun fiyatlara bulabileceğimiz AVR prototipleme kartı. Üzerinde programlayıcı bulundurmadığı için ayrı bir programlayıcıya ihtiyaç duyuracaktır. Aynı zamanda pek çok modelinde güç soketinde adaptör yuvası vardır. Resim : https://megaeshop.pk/media/catalog/product/-/f/-font-b-atmega8-b-font-atmega48-development-board-font-b-avr-b-font-development-board_1_.jpg

Arduino Uno Geliştirme Kartı

Geliştirme ve deneme kartları arasında tercihe değer olarak Arduino Uno kartını bulduk. Arduino kart kullanılması kesinlikle Arduino'nun yazılımını kullanacağımızı akla getirmesin. Arduino burada sadece fiziksel kart ve ön yükleyici olarak kalacak. Geri kalan tüm kodlar saf dile uygun olarak yazılacak. Donanım olarak Arduino'nun tercih edilmesinin pek çok avantajı vardır. Bunları madde madde sıralayabiliriz,

- Arduino UNO kartı harici bir beslemeye ve besleme devresine ihtiyaç duymaz. Üzerinde 5V ve 3.3V regülatörler bulunur ve programlayıcı, mikrodenetleyici ve dış elemanlar buradan beslenir.
- Arduino UNO kartının beslemesi USB üzerinden yapılabilir. Böylelikle DC adaptör kullanmaya ihtiyaç kalmaz.
- Arduino UNO kartı harici bir programcıya ihtiyaç duymaz. Kendi programcısı üzerinde gelir.
- Arduino UNO kartı bilgisayar ile üzerinde bulundurduğu çevirici entegre ile USB üzerinden doğrudan iletişime geçebilir.
- Arduino UNO kartında kısa devre ve akım koruması vardır.
- Arduino UNO kartı için üretilmiş üstlükleri (shield) kullanma imkanımız vardır.



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Arduino Uno Ayak Haritası Resim: https://components101.com/sites/default/files/component_pin/Arduino-Uno-Pin-Diagram.png

Derslerimizde yettiği kadarıyla Arduino UNO kartını kullanacağız. O yüzden bu derslerle AVR programlamayı öğrenmek isteyenlerin Arduino UNO kartını tercih etmesi gerekir. İleri derslerde yazılım, programcı seçiminden bahsedip Arduino Uno kartının donanımını anlatarak devam edeceğiz.

Programlayıcı Seçimi

Programlayıcılar bir mikrodenetleyici platformunun olmazsa olmazlarıdır. Programlayıcıya erişim olmadığı sürece elimizdeki mikrodenetleyicilere program atmamız mümkün olmaz. O yüzden bir mikrodenetleyici platformuna giriş yapmadan önce uygun programlayıcıyı temin etmek gerekir. Ülkemizde Atmel'in resmi programlayıcılarını elektronik sitelerinde pek göremesem de klon programlayıcıları rahatça bulabiliyorum. Ayrıca Aliexpress'de orjinal olmasa da uygun fiyata klon programlayıcıları temin etmek mümkün. Programlayıcının yanında hata ayıklama özelliği (debugger) olan cihazlar da vardır. Bunların klonu olmadığı gibi fiyatları da normal programlayıcılara göre oldukça yüksektir. Hata ayıklama özelliğinin olması programcıya büyük kolaylık sağlasa da başlangıçta bu özellikten feragat edebiliriz. Klonu çıkmış bir hata ayıklayıcı olsa da Atmel Studio 4 ile kullanılabildiğinden ondan bahsetme gereği duymuyoruz. Şimdilik iki klon programlayıcıyı tanıtmakla yetinelim.

usbASP

En uygun fiyatlı AVR programlayıcısı olup rahatlıkla bulunabilir. Üçüncü parti bir programlayıcı olsa da çoğu yazılım ile uyumludur. Atmel Studio ile doğrudan bir uyumluluğu olmasa da sonradan araç olarak eklenebilir. Bazı modellerinde 3.3V-5V arası seçim yapan bir anahtar vardır. Çoğunda ise 5 voltta çalışmaktadır. 5 voltta çalışması 3.3 voltluk devreler için sıkıntı doğurabilir. Gücünü USB'den alır ve bağlandığı devreye doğrudan bu beslemeyi sağlar. İstenirse PCB görüntüsü alınıp elde de yapılabilir. Devresi paylaşıldığından pek çok üretici kendi programlayıcılarını üretmiştir.

usbASP sürücülerine ve devre şemasına şu bağlantıdan ulaşabilirsiniz.

https://www.fischl.de/usbasp/

Resim: https://ae01.alicdn.com/kf/HTB1rtGASpXXXXayapXXq6xXFXXXG/New-USBASP-USBISP-AVR-Programmer-Free-Driver-USBASP-USBISP-AVR-51-AVR-ISP-Downloader-Programmer-for.jpg_640x640.jpg

AVRISP mkII

Atmel'in resmi programlayıcısı olan bu cihazın klonları üretilmiştir. usbASP kadar uygun olmasa da yine de makul fiyatlara bulabileceğiniz bu cihaz Atmel Studio ile beraber kullanılabilir. usbASP'ye göre içinde pek çok özellik barındırır. Hata ayıklama özelliği olmasa da devrenin gerilimini otomatik ölçen sistemi, devreden ayrı beslemesi, flash ve eeprom programlayabilmesi, kısa devre koruması vardır.

Resim: https://www.microchip.com/_ImagedCopy/ATAVRISP2.jpg

İleri seviye bir programcı ve hata ayıklayıcı arayan Atmel ICE ürününe göz atabilir. Biz şimdilik hobiciler ve öğrenciler tarafından alınabilecek olan programcılardan bahsetmekle yetinelim. Bir sonraki konumuzda gerekli ve tavsiye ettiğimiz yazılımlardan bahsedeceğiz.

Kaynaklar: AVRISP MkII, Microchip, https://www.microchip.com/developmenttools/ProductDetails/atavrisp2, Erişim Tarihi: 15.08.2015 usbASP, Thomas Fischl, https://www.fischl.de/usbasp/, Erişim Tarihi: 15.08.2015

Yazılım

Bundan önceki dört derste tamamen donanım yönünden AVR'ye giriş yaptık. Şimdi işin yazılım boyutunu ele alıyoruz. Bu yazılımlar arasında geliştirme stüdyosu, derleyici, programcı, metin editörü gibi gerekli ya da faydalı yazılımlar var. Bu yazılımların tamamını ücretsiz olarak indirip kullanabiliriz. Atmel'in en büyük özelliklerinden biri de ücretsiz geliştirme ortamı sunmasıdır. Visual Studio tabanlı bu geliştirme ortamı AVR Assembly, C ve C++ dillerinde yazılım geliştirme imkanı sunmaktadır.

Sadece Assembly değil de AVR Assembly dememizin bir sebebi var. Çünkü Assembly dilinde makine kodu alfanümerik (harf ve rakam) sembollere çevrilmiştir. Bu yüzden her mimarinin kendine ait bir assembly dili vardır. x86 assembly, z80 assembly, PIC assembly gibi ayrı diller olarak zikretmek gerekir. Tek bir assembly dili yoktur.

Atmel Studio

Atmel Studio, Atmel'in resmi olarak yayınladığı geliştirme stüdyosudur. Günümüzde 6.2 ve 7 versiyonları kullanılsa da biz 6.2 versiyonunu kullanmaya devam edeceğiz. Dileyen 7. sürümünü kullanabilir arasında fazla fark yoktur. Atmel Studio'yu aşağıdaki bağlantılardan indirip kurabilirsiniz.

http://www.microchip.com/mplab/avr-support/avr-and-sam-downloads-archive

CH340 Sürücüsü

Klon Arduino UNO kartında USB-TTL çevirici olarak CH340 entegresi bulunur. Arduino IDE yüklense dahi bu sürücü resmi olarak tanınmadığı için ayrıca yüklemek gerekir. Bunun için aşağıdaki bağlantıdan sürücüyü indirip yükleyebilirsiniz.
https://sparks.gogo.co.nz/assets/_site_/downloads/CH34x_Install_Windows_v3_4.zip

AVRDUDE

Atmel Studio'da Arduino UNO'yu kullanabilmek için AVRDUDE programına ihtiyaç vardır. Bu programı C:\AVRDUDE adresine çıkarıyoruz.
http://download.savannah.gnu.org/releases/avrdude/avrdude-5.11-Patch7610-win32.zip

Arduino

Bazen Arduino yazılımlarını bazen de kaynak kodunu AVR üzerinden inceleyebileceğimiz için Arduino derleyicisini indirmekte fayda vardır. Aşağıdaki bağlantıdan indirebilirsiniz.

https://www.arduino.cc/en/Main/Software

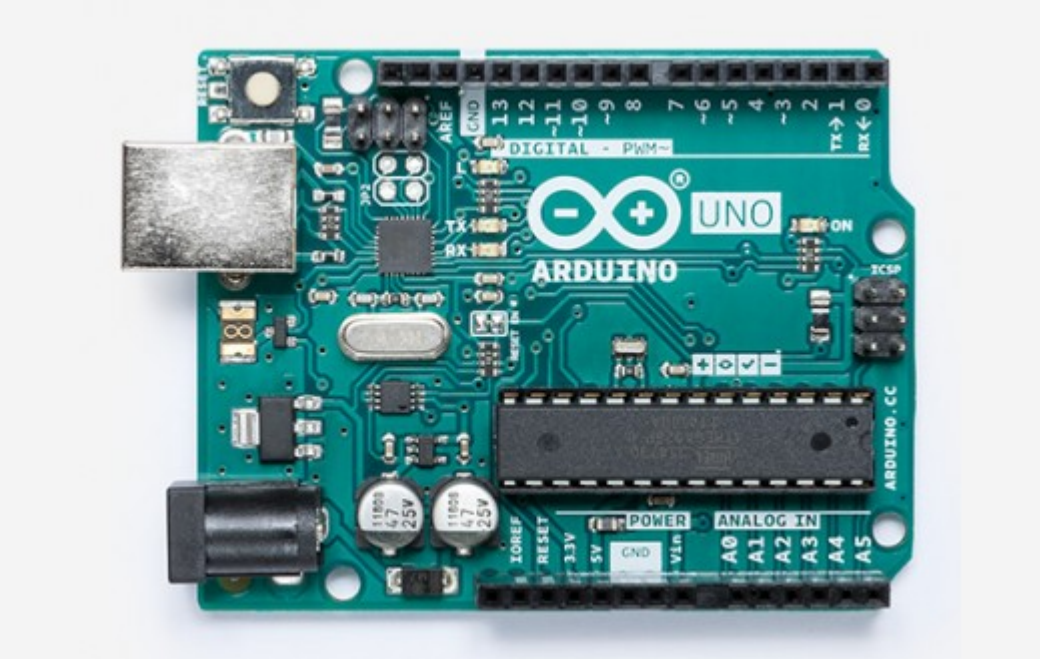
Notepad ++

Kod incelemede ve kod yazmada faydası olduğu için sıkça kullanacağımız bir kelime işlem programı olup ücretsiz olarak şu bağlantıdan indirilebilir.

https://notepad-plus-plus.org/download/v7.5.8.html

İleride gerekli başka bir program olduğunda gerektiği zaman bunu dile getireceğiz. Başlangıç için gerekli yazılımlar bu kadarla sınırlıdır.

Arduino UNO Kartını Tanıyalım

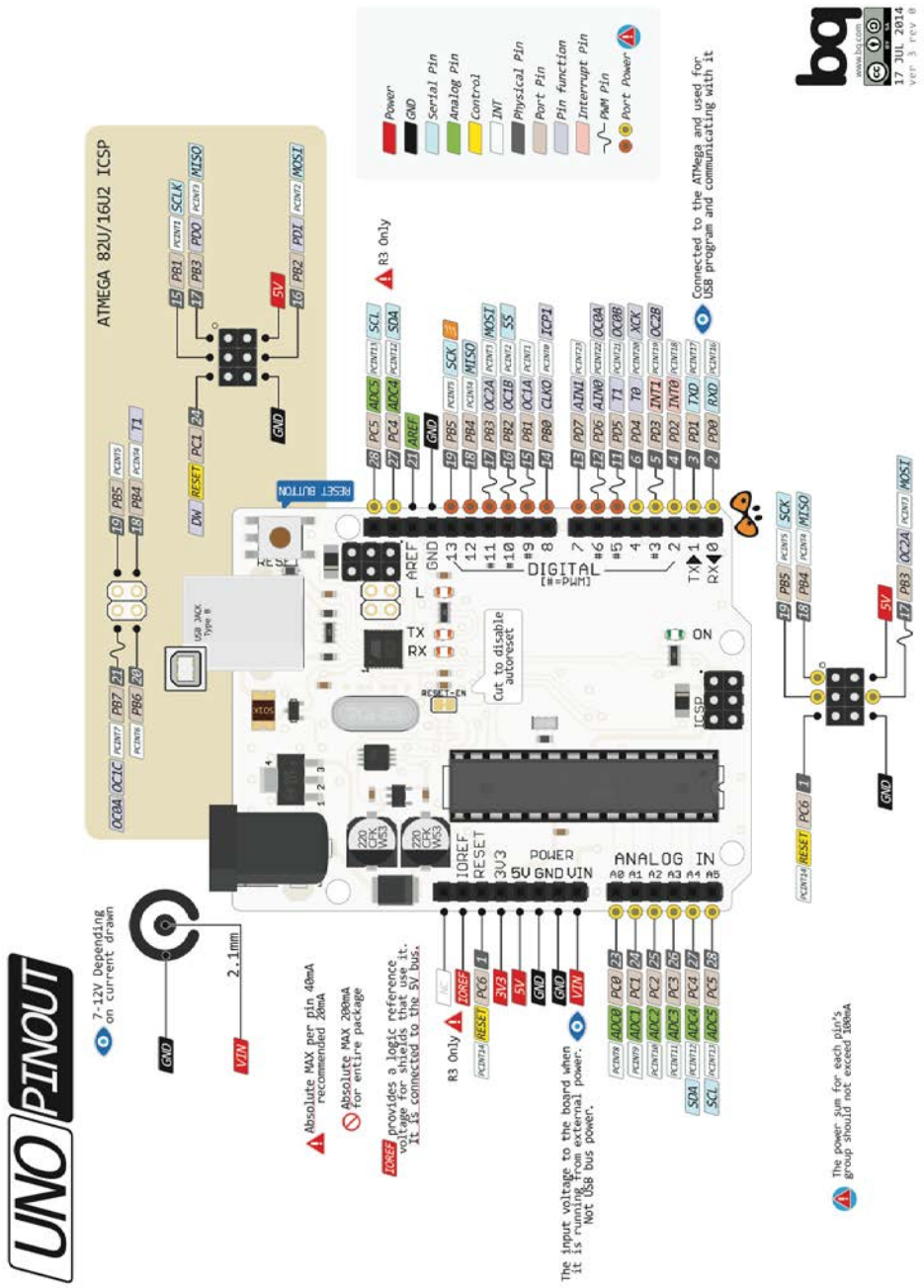


Resim : https://store-cdn.arduino.cc/usa/catalog/product/cache/1/image/520×330/604a3538c15e081937dbfbd20aa60aad/a/0/a000066_featured_4.jpg

Arduino UNO kartı Arduino'yu Arduino yapan karttır diyebiliriz. Arduino'nun en sık kullanılan kartı olup Arduino dediğimizde akla ilk gelen karttır. Önce RS-232 portlu ve DIP soketli versiyonları çıkmış zamanla USB özellikli ve SMD yapıya dönüşmüştür. Tasarımda zaman içerisinde gelen bu değişim sadece belli kısımlarda olmuş bazı çizgiler hep aynı kalmıştır. O yüzden yıllar önce yazılmış bir kod bile devrede ya da kodda bir değişiklik olmadan kullanılabilir. Arduino için üretilmiş olan üstlüklerin hemen hepsi Arduino Uno kartına göre üretilmektedir. Arduino'nun en yaygın ve pratik kartı olan UNO'yu biz sadece donanımı için tercih edeceğiz. Aşağıdaki tabloda UNO kartının teknik özellikleri verilmiştir.

Mikrodenetleyici	Atmel Atmega328P
Çalışma Gerilimi	5V
Giriş Gerilimi	7-12V (6-20V MAX)
Dijital Giriş ve Çıkış Ayakları	14 (6 adet PWM)
Analog Giriş Ayakları	6
Giriş Çıkış Ayağı Akımı	20mA
Program Hafızası	32KB (0.5 KB Ön yükleyici)
SRAM	2KB
EEPROM	1KB
Saat Hızı	16MHz

AVR geliştiricisi olarak Arduino UNO kartındaki numaralandırılmış dijital giriş ve çıkış ve analog girişlerden değil mikroişlemcinin ayak numaraları bizi ilgilendirdiği için aşağıdaki diyagrama bol bol başvurmamız gerekecek.



Resim: http://forum.arduino.cc/index.php?action=dlattach;topic=146315.0;attach=90365

Şimdi diyagramdan anladıklarımızı aşağıda özetleyelim.

Güç Kısmı

Power diye adlandırılan kısımda 5V, IOREF (5V ile aynı), 3.3V regülatöre bağlı 3.3V, Şase ve VIN yani gerilim girişi bulunur. VIN kısmı 5V regülatöre bağlıdır. Eğer USB'den besleme sağlanıyorsa buradan 5V elde edilebilir. Reset kısmı mikrodenetleyicinin reset ayağına aynı zamanda da reset düğmesine bağlıdır. Harici bir resetleme için buradan bağlantı yapılabilir.

Analog Kısmı

Analog kısmı C portuna bağlıdır. Burada A4 ve A5 TWI (I2C) iletişim için kullanılır.

Dijital Kısmı

Burada aşağıdan başlayarak UART, Kesme, Zamanlayıcı, SPI, Analog Referans ve LED ayakları bulunur.

Analog ve Dijital kısımdaki ayakların tamamı dijital giriş ve çıkış olarak kullanılabilir. Biz programlarken bunu B portu, C portu, D portu olarak programlayacağız. Dijital ayak numaralarıyla bir işimiz olmayacak. Karta şöyle bir baktığımızda mikrodenetleyicinin ayaklarını kullanıma sunan header soketleriyle mikrodenetleyici ayakları arasında bağlantı kuran ve gerekli güç ve program devresini hazır olarak üzerinde bulunduran bir kart olarak karşımıza çıkıyor.

Arduino UNO'nun şemasını şu bağlantıdan inceleyebilirsiniz,
https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

Kaynaklar:

DÖKMETAŞ,Gökhan,"Arduino Eğitim Kitabı", İstanbul,2016 Arduino – ArduinoUno, arduino.cc, https://www.arduino.cc/en/Guide/ArduinoUno, Erişim Tarihi : 15.08.2018

İlk Program

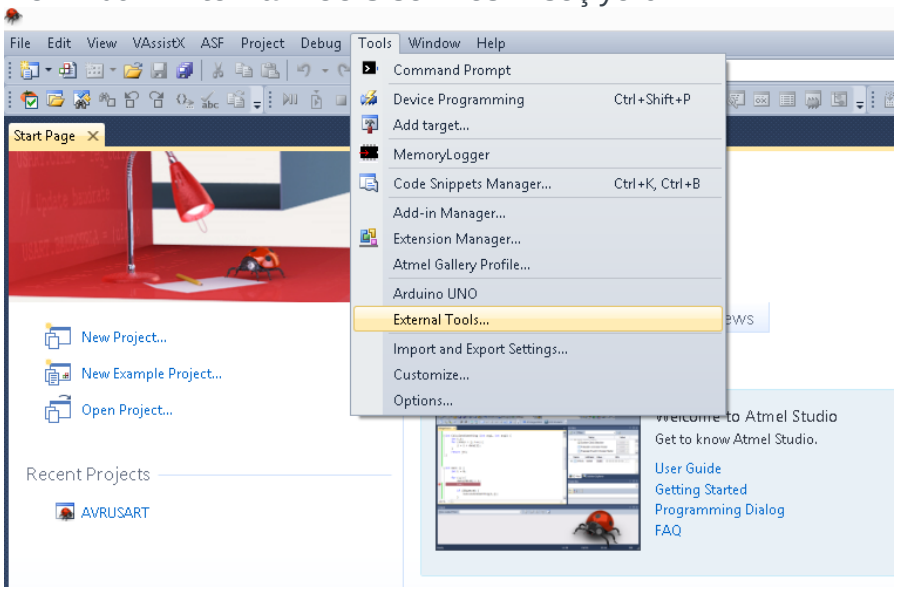
Bundan önceki 6 derste AVR'yi tanıtıp sizi AVR'ye hazırladık. Şimdi ilk programı işletme vakti. Fakat söylememiz gereken çok fazla teorik bilgi olduğu için sadece bir program yapıp sonra teorik bilgileri anlatmakla devam edeceğiz. Teorik bilgilerin bolluğu bu seviyede pek sorun olmayacaktır. Eğer mikrodenetleyici dünyasına yeni girdiyseniz ve pratik eksikliğiniz varsa daha basitleştirilmiş ve yani başlayan dostu olan Arduino'yu deneyebilirsiniz. Yazdığım Arduino Eğitim Kitabı sıfırdan başlayanları AVR ile ilgili konuları anlayacak seviyeye getirecek düzeydedir. Sağlam bir temel oluşturmadan ileri seviye konuların anlamadan ve ezbere yapılma olasılığı çok fazladır. Ezberciliğin önüne geçmek için mümkün olduğu kadar ince ayrıntısıyla ve herkesin anlayacağı düzeyde açıklayarak fakat bir birikimi olan okuyucuları da sıkmayacak seviyede anlatmaya gayret edeceğim.

C ile AVR programlama mantığını anlamadan oldukça anlaşılmaz ve zor, mantığını anlayarak oldukça anlaşılır ve kolay gelecektir. C dilinde programlamamızın yanında elimizde belli sınırları ve özellikleri olan bir işlemci vardır. Portlar, Kesmeler, Zamanlayıcılar, İletişim, Sigortalar, ADC, PWM gibi konuları öğrendikten sonra geriye fazla

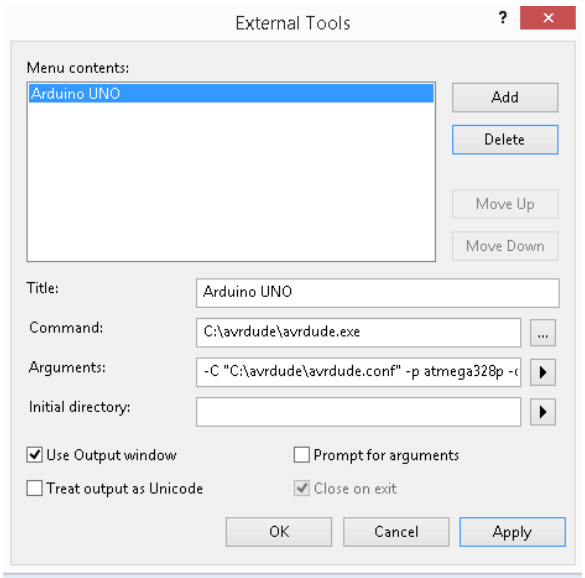
bir şey kalmamaktadır. Bu işlemler de genel olarak belli yazmaçların bit değerlerini okuma, bu yazmaçlara bit değeri yazma ve bu bit değerlerini işlemcide işleme veya kaydetme olarak yapılmaktadır. Çevre birimleriyle ve dış dünya ile iletişimi çözdükten sonra geri kalan işimiz yazılımsal boyutta olacaktır.

Arduino UNO'yu Atmel Studio'da Kullanmak

Arduino UNO kartı Atmel Studio ile doğrudan kullanılamaz. Bunun için Atmel Studio'nun bir özelliği olan **Tool** (Alet) olarak tanımlanması gerekir. Fakat ondan da önce AVRDUDE programını yüklememiz gerekir. Bundan öncesinde AVRDUDE programından bahsetmiştik. Okumayanlar buradan okuyabilir. C sürücüsünde AVRDUDE adında bir klasör açıp indirdiğimiz arşiv dosyasını oraya çıkardıktan sonra Atmel Studio'yu açıyoruz. **Tools** kısmından **External Tools** sekmesini seçiyoruz.



Burada açılan pencerede ADD düğmesine tıklayıp yeni bir alet oluşturuyoruz. Bilgiler resimdeki gibi girilmelidir.



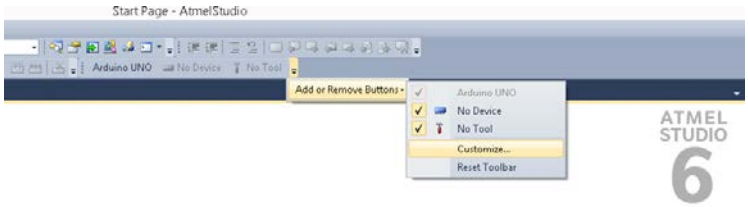
Title	Arduino UNO (veya istediğiniz bir başlık)
Command	**C:\avrdude\avrdude.exe**
Arguments	**-C “C:\avrdude\avrdude.conf” -p atmega328p -c arduino -P COM9 -b 115200 -U flash:w:”$(ProjectDir)Debug\$(ItemFileName).hex”:i**

Use Output Window kutusunu işaretli değilse işaretliyoruz.

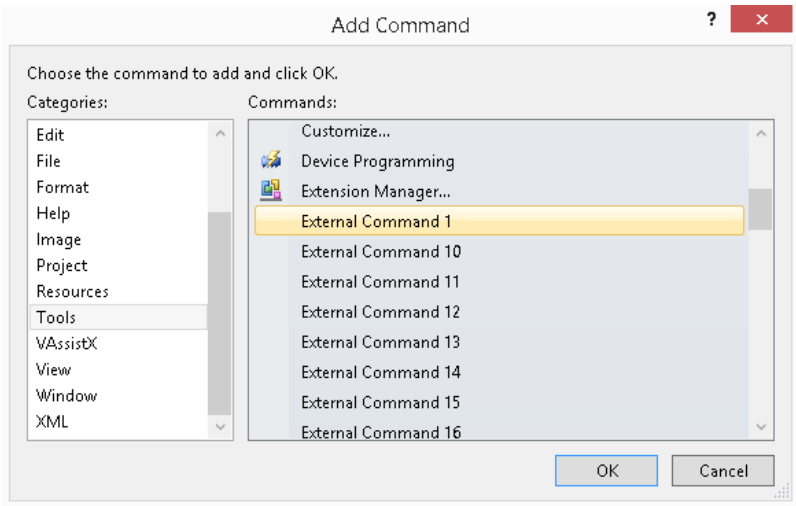
COM9 yerine Aygıt Yöneticisinden baktığımız COM değerini yazıyoruz.

Özellikle Arguments kısmındaki komutun doğru olarak yazıldığından emin olun. AVRDUDE programına gönderilecek bilgiler burada yer alır ve Arduino UNO’ya göre düzenlenmiştir. Eğer farklı bir kart ya da entegre kullanmayı istiyorsanız argümanların değerlerini değiştirerek programı işletebilirsiniz. Burada Atmel Studio AVRDUDE programı üzerinden Arduino kartına debug klasöründeki .hex dosyasını atacaktır.

Bu işlem ile beraber Atmel Studioda derleyeceğimiz program doğrudan Arduino UNO kartına atılacaktır. Ama bunun kısayolunu oluşturup ekrandaki bir düğmeye basarak işi kolaylaştıralım. Öncelikle şekildeki gibi bir düğme oluşturmak için sağ taraftaki oka tıklayıp Add or Remove Buttons sekmesinden Customize düğmesine tıklıyoruz.

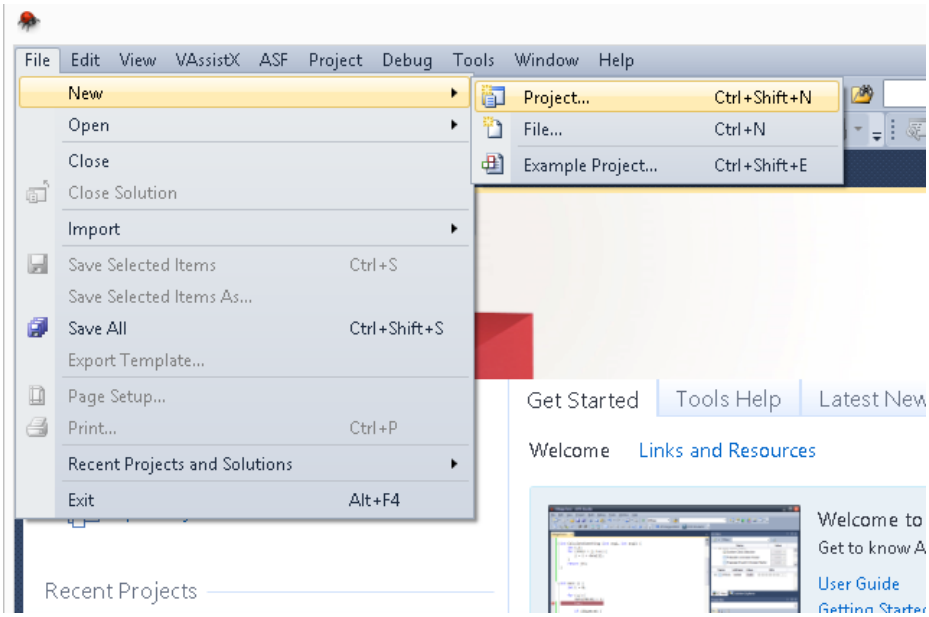


Açılan pencerede **Add Command…** düğmesine tıklıyoruz ve Tools sekmesinden External Command 1’i seçiyoruz.

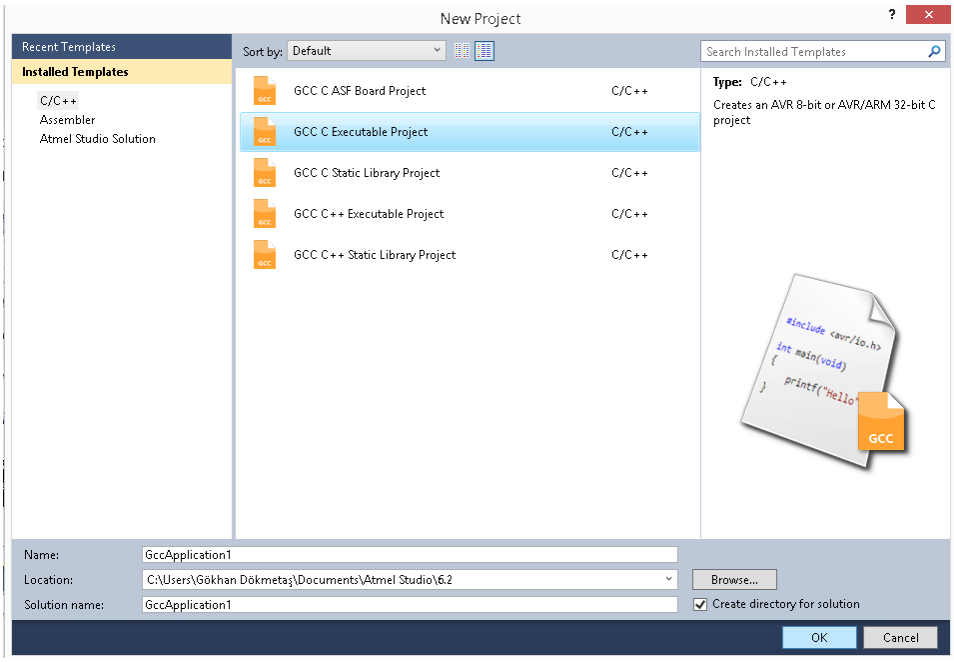


Böylelikle şimdilik Atmel Studio'da yapacağımız ek bir şey kalmamış oluyor. Şimdi yeni proje açalım ve ilk programı yazıp çalıştıralım.

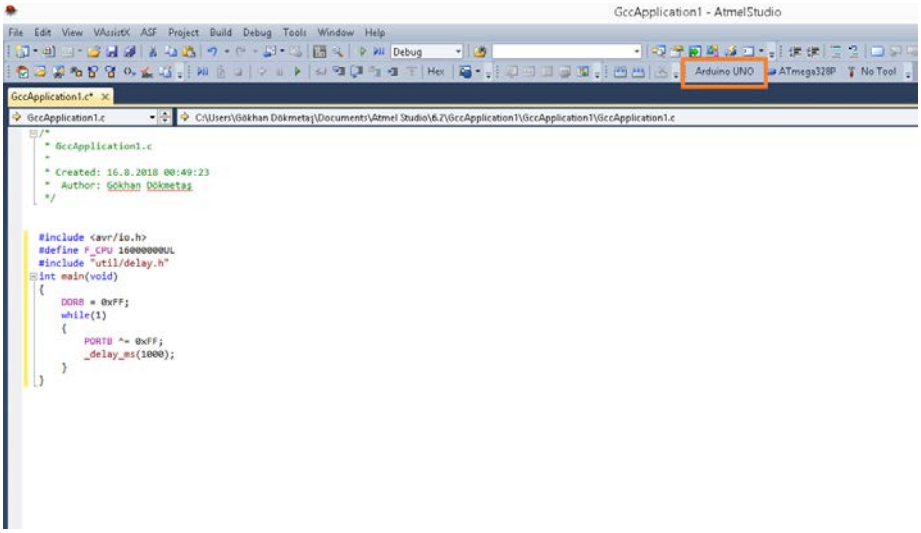
Öncelikle File/New/ New Project diyoruz.



Karşımıza iki dil ve o dillerdeki proje tipleri çıkıyor. Assembly ve C/C++ dilleri ayrı olduğu gibi C ve C++ projeleri de ayrı olarak açılmaktadır. GCC C Executable Project seçeneğini seçiyoruz ve projemize isim verdikten sonra projeyi açıyoruz.



Proje ekranından sonra karşımıza aygıt seçme ekranı geliyor. Atmega328P kullanacağımız için arama kısmına Atmega328P yazıyoruz. Ayrıca aygıtı geçtikten sonra teknik veri sayfası (datasheet) ve desteklenen araç listesi çıkıyor. Buranın ekran görüntüsünü verme gereği duymadım çünkü yukarıdaki ekran görüntüleri yeteri kadar fazla oldu. Şimdi karşımıza programı yazacağımız .c uzantılı dosya geliyor ve programı yüklemek için yukarıdaki Arduino UNO düğmesine tıklıyoruz.



Şimdilik Atmel Studio'yu kullanmak hakkında vereceğim bilgiler bu kadarı ile sınırlı olacak. Şimdi basit bir yanıp sönen led programı ile Arduino UNO'nun üzerindeki L etiketli ledi yakıp söndürelim.

```c
/*
 * GccApplication1.c
 *
 * Created: 16.8.2018 00:49:23
 *  Author: Gökhan Dökmetaş
 */


#include <avr/io.h>
#define F_CPU 16000000UL
#include "util/delay.h"
int main(void)
{
	DDRB = 0xFF;
	while(1)
	{
		PORTB ^= 0xFF;
		_delay_ms(1000);
	}
}
```

Programın açıklamasını burada yapmayacağız. Sadece deneme amaçlı bir program olarak burada bırakalım. İlerleyen derslerde PORT yazmaçlarını ve port manipulasyonunu ayrıntılarıyla açıklayacağız.

Kaynaklar: Using Arduino Boards in Atmel Studio, Sepehr Naimi, http://www.microdigitaled.com/AVR/Hardware/Arduino/UsingArduinoBoardsInAtmelStudio.pdf, Erişim Tarihi : 16.08.2018

Atmega328P Entegresini Tanıyalım

Arduino UNO kartı üzerinde bulunan Atmega328P dersler boyunca kullanacağımız ve onun üzerinden dersleri anlatacağımız mikrodenetleyici olacaktır. O yüzden programlamaya girmeden önce mikrodenetleyiciyi teknik veri kitapçığından (datasheet) faydalanarak anlatalım. Her entegre devrenin üretici tarafından yayınlanan ve bize lazım olacak her bilgiyi içeren teknik veri kitapçığı vardır. Bir mikrodenetleyiciyi veya işlemciyi programlamak için bu teknik veri kitapçığına ihtiyaç duyarız. Assembly dilinde programlama yapmayacağımız için komut seti kitapçığına ve diğer alt seviye bilgiye ihtiyacımız olmasa da yazmaçlar ve bunların özelliklerini muhakkak öğrenmemiz lazımdır. C dilinde programlama yapmamız bu kitapçıktaki bilgilerin tamamını öğrenmek zorunda bırakmıyor fakat büyük bir kısmını yine de öğrenmemiz gerekli. Teknik veri kitapçıklarının neredeyse tamamı İngilizce yazılmaktadır ve bunu geliştiricinin okuyup anlaması gereklidir. Biz burada işi biraz kolaylaştırarak teknik veri kitapçığında gerekli yerleri Türkçe olarak anlatmaya çalışacağız. Böylelikle dil engelini elimizden geldiği kadarıyla ortadan kaldıracağız.

Atmega328 Özellikleri

- 131 işlemci komutu
- Çoğu komut tek çevirimde çalışır. (Single cycle)
- 32×8 Genel kullanım yazmacı
- 20MHz'e kadar saat hızı
- Entegre 2 çevirim çarpıcı (multiplier)
- 32KB Programlanabilir FLASH program hafızası
- 1KB EEPROM
- 2KB Dahili SRAM Bellek
- 10.000 Flash / 100.000 EEPROM okuma-yazma kapasitesi
- 85 derecede 20 yıl, 25 derecede 100 yıl veri koruma.
- Kilit Bitleri ve Kod koruma
- İki adet 8-bit zamanlayıcı ve sayıcı ölçeklendirme ve karşılaştırma modlarıyla.
- Bir adet 16-bit zamanlayıcı ve sayıcı ölçeklendirme, yakalama ve karşılaştırma modlarıyla.
- Gerçek zaman sayıcı ayrı osilatör ile
- Altı PWM kanalı
- 6 kanal 10-bit ADC, Sıcaklık ölçümü
- İki adet ana/uydu SPI Seri Arayüz
- Bir adet programlanabilir Seri USART
- Bir I2C arayüzü
- Programlanabilir WDT
- Bir Analog Karşılaştırıcı
- Ayağa bağlı uyandırma ve kesme
- Açılışta reset ve kararma (brown-out) saptaması
- Dahili ayarlı osilatör
- Harici ve Dahili Kesme Kaynakları
- Altı uyku modu
- 23 programlanabilir giriş ve çıkış ayağı
- 1.8 – 5.5 V arası çalışma gerilimi
- -40 ve +105C arası çalışma sıcaklığı
- Aktif modda 0.2mA Power-down modunda 0.1uA
- Power-save modunda 0.75uA
- Güç tüketimi. (Buna diğer unsurlar dahil değildir.)

Atmega328p mikrodenetleyicisinin özellikleri aşağı yukarı yukarıdaki kadardır. Bazı özellikler çok kullanılan özellikler olmayıp bazıları da muhakkak bilinmesi gereken özelliklerdir. Dersler boyu konuların önemine göre her konuya farklı ağırlık vereceğiz. Şimdi yukarıdaki özelliklerin bir kısmını açıklayalım.

İşlemci Komutu: İngilizcesi "Instruction Set" olan bu tabir işlemcinin kaç komutla çalıştığını ifade eder. Daha fazla komut sayısı öğrenmeyi zorlaştırsa da komut fazlalığından dolayı işleri daha az komutla yapmayı sağlar. Az komut ise öğrenmesi daha kolay fakat karmaşık işlerde komut fazlalığı ve yazılması daha zor bir program gibidir. 1000 kelimelik bir dil ile 10000 kelimelik bir dil gibi düşünebiliriz bu durumu. Biz C dilinde programlama yapacağımız için işlemci komutlarıyla pek işimiz olmayacak. Eğer Assembly dili üzerinde çalışsaydık bunları öğrenmemiz gerekecekti.

Genel Kullanım Yazmacı

İşlemci ile RAM bellek arasında bilgisayarlarda Cache bellek olduğu gibi mikrodenetleyicilerde de yazmaçlar vardır. Bu yazmaçların bazıları genel kullanıma ayrılmıştır. Assembly dilinde programlama yapanlar bu yazmaçlar üzerinden verileri alıp işleyip kaydederek programlamayı yaparlar.

Saat Hızı

İşlemcinin çalışma hızı demektir. Fakat bu işlem hızı anlamına gelmez. PIC mikrodenetleyiciler bir komutu işlemek için 4 çevirime ihtiyaç duyar. İşlem hızı da saat hızının dörtte birine denk gelir. AVR'de bu genellikle 1/1 oranındadır. Mikrodenetleyiciye bağlayacağımız osilatöre göre saat hızını belirleriz.

Flash Program Hafızası

Bilgisayarda yazdığımız ve derlenip .hex dosyasına dönüşen kodun yüklendiği hafızanın adıdır. Sadece program yüklenebilmektedir ve çalışma esnasında bir erişim söz konusu değildir. Bu aynı boş bir CD'ye veri yazıp kullanmak gibidir. Programlama esnasında program hafızasına program kodundan başka değerler de yazabilsek de çalışma esnasında bu değerler sadece okunabilir hale gelir. Atmega328P'de ön yükleyici özelliği sayesinde program hafızasının bir kısmını silinmeden diğer bir kısmına silme ve yükleme işlemi yapılabilir.

SRAM

Bilgisayarlardaki RAM bellek ile aynı görevi görür. Bizim algılayıcılardan, kullanıcıdan aldığımız değer bu belleğe atılır. Ayrıca değişkenlerin değeri de RAM bellekte tutulur. RAM bellekteki değerler elektrik kesildiğinde silinmektedir. Çalışma esnasında gerekli olan değerler burada tutulur. Ram bellekler ROM bellekler gibi belli bir okuma ve yazma ömrüne sahip olmadığı için sürekli okuma ve yazma yapılan çalışma esnasında RAM bellek kullanılır.

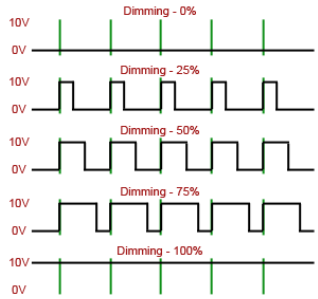
EEPROM

Program hafızasından ayrı olarak kalıcı verilerin saklandığı bellektir. Bu belleğin özelliği önceden programlamaya ihtiyaç duyulmadan çalışma esnasında kalıcı verilerin saklanmasına imkan sağlamasıdır. Program hafızasına çalışma esnasında kayıt olmadığı için mikrodenetleyici ek bir kalıcı belleğe ihtiyaç duyar. EEPROM bu ihtiyaç sonucu mikrodenetleyicilere eklenmiştir. Örneğin ayar değerleri, şifre, ip adresi gibi veriler kullanıcı tarafından girilir ya da değiştirilir ve bunlar EEPROM'da saklanır.

Zamanlayıcı ve Sayıcı

Bunu ayrı zikretmemiz bunları ayrı birer birim olduğu izlenimi vermesin. Mikrodenetleyicinin iç saatine göre sayma işlemi yapıldığında zamanlayıcı, dış kaynaktan sayma yapıldığında ise sayıcı olarak adlandırılır. Mikrodenetleyici içerisinde bir saatin olması zamana bağlı işlerde merkezi işlem birimini (CPU) meşgul etmeden harici olarak sayma işlemini yapmaya imkan verir. Mikroişlemciye bekleme komutu verip belli bir zaman kullanılamaz hale getirmek yerine zamanlayıcıdan belli aralıklarla okuma yapıp o değere göre işlemi yapmak zamana bağlı işlemlerde mikrodenetleyiciyi gereksiz meşguliyetten kurtaracaktır. Sayıcı görevinde ise dışarıdan gelen frekansa göre değer artar ve belli bir süre sonra taşma yapar. Bit değeri kapasite bakımından önemlidir. 8-bit bir sayıcıda sadece 255'e kadar sayma işlemi yapılmaktadır o yüzden çok sık aralıklarla kontrol edilip sıfırlanması gerekebilir. 16-bit zamanlayıcı bizi bu konuda biraz daha rahat bırakmaktadır.

PWM PWM (Pulse-Width Modulation) dijital devrelerin sadece HIGH ya da LOW değeri vermesinden dolayı getirilmiş bir çözümdür. Dijital devrelerin analog gibi çıkış vermesini sağlar. Bu analog gibi çıkış belli bir frekanstaki kare dalganın yüksek kenarının genişliğinin miktarı ile belirlenir. Aşağıdaki resimde PWM'nin çalışma prensibine bir örnek verilmiştir.



Resim: https://qph.fs.quoracdn.net/main-qimg-284e6cd1504b56902da50778cd233c9a

ADC

Analog-Digital Converter ifadesinin kısaltılmış halidir. Yani Analog değeri okuyup dijital veriye dönüştüren birimdir. Analog değer mikrodenetleyicinin çalışma gerilimi ve 0V arası ya da referans değeri ile 0V arası olabilir. Bu değer doğru orantıya çok yakın olarak dijital değere çevrilir. 0-5V arası 10-bitlik bir dönüşümde 0-1023 arası değerlere dönüştürülür.

İletişim protokollerine geldiğimizde iletişim protokollerinden ayrıntılı olarak bahsedeceğimiz için şimdilik burada bırakalım. Bir sonraki yazıda görüşmek üzere.

Kaynaklar ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 16.08.2018

Port Yazmaçları

Bu derste geride kalan 8 dersin anlattığının tamamından daha önemli ve zor bir konuya giriş yapacağız. Artık kolay konuları geride bıraktık diyebilirim. Bu konu en önemli konulardan biri olduğu için tamamen anlaşılmadıkça diğer konulara geçilmemesi gerekir. AVR'de en sık yapacağımız giriş ve çıkış işlemleri aynı zamanda ilk yapacağımız işlem olacaktır. Önceki derste led yakma uygulaması örneğinde olduğu gibi temel giriş ve çıkışların kullanılmadığı uygulama yok gibidir. En basit uygulamadan en karmaşık uygulamaya kadar kullanmak zorunda kalacağımız temel giriş ve çıkış birimlerini anlamakla aynı zamanda yazmaçlar üzerinde işlem yapmayı da anlayıp diğer pek çok konuyu rahatça anlayabilirsiniz.

Yazmaç (Register) Nedir?

Sürekli bahsettiğimiz ama şimdiye kadar ayrıntılı olarak açıklamadığımız yazmaçlar küçük veri hücreleridir. Bu hücreler mikroişlemcinin parçalarından olup gerekli ayar, okuma, yazma ve çalışma verilerini içinde barındırır. Küçük veri hücrelerine yazılan bilgi mikrodenetleyicinin nasıl çalışacağını belirler. Aynı zamanda bu hücrelerden okunan bilgiler ile mikrodenetleyicinin işlemcisi işlem yapar. Mikrodenetleyicinin ayarlarını bu hücrelerdeki verilerin konumlarını değiştirerek yaparız. 8-bit mikrodenetleyicide 8-bitlik yazmaçlar bulunur. Bir yazmacın örnek bir görüntüsü aşağıdaki tablodaki gibidir,

0	0	1	0	0	1	0	1

Burada bir (1) ve sıfır (0) olarak temsil edilen rakamlar aslında gerçekte elektriğin var olup olmamasıdır. Elektrik tabanlı olmayan ROM, CD gibi aygıtlarda ise iki durumdan birinin karşılığıdır. Anlaşılması için biz 1 ve 0 olarak sayılarla ifade ediyoruz. bir (1) HIGH ve TRUE, sıfır (0) ise LOW ve FALSE olarak adlandırılır.

AVR mikrodenetleyicilerde bulunan bazı yazmaçların bazı bitleri belli ayarları yapmaya, bazı bitleri belli bilgileri okumaya, bazı yazmaçların kendisi veri tutmaya ayarlanmıştır. Bu yazmaçların ve bitlerinin görevleri teknik veri kitapçığında açıklanmıştır. Yazmaçların bitleri sağdan itibaren 0-7 arası rakamlandırılır. Programcı bir yazmacın değerini okur, yazmaca değer yazar, yazmacın bir bitindeki değeri değiştirir, yazmacın bir bitindeki değeri okur ve buna göre programlama yapar. Bu yazmaçlar mikrodenetleyicinin çevre birimleri ile irtibatını, ayarlarını, denetimini sağlar. Yazmaçların bitlerine müdahaleyi C'nin bitwise operatörleri ile sağlarız. Genelde pek üzerinde durulmayan bitwise operatörler gömülü sistemlerde en önemli operatörlerden biridir.

Mikrodenetleyicinin teknik veri kitapçığına baktığımızda giriş ve çıkış ile alakalı üç ana yazmaç ve bununla alakalı bir ayar bitini bulunduran bir yazmaç karşımıza çıkıyor. Öncelikle ayar bitini bulunduran yazmacı açıklayalım. Bu yazmacı açıklamakla yazmaç mantığını biraz daha hızlı öğrenmenizi umuyoruz.

MCU Control Register (Mikrodenetleyici Denetim Yazmacı)

Yazmacın Biti	7	6	5	4	3	2	1	0
Bit Adı	— BOŞ—	BODS	BODSE	PUD	— BOŞ—	— BOŞ—	IVSEL	IVCE
Erişim		Y/O	Y/O	Y/O			Y/O	Y/O

Bu yazmaç mikrodenetleyicinin denetimi ile ilgili 5 adet ayarı içinde barındırır. Bu bitleri açık ya da kapalı yaparak ayarlamayı yapabiliriz. Her konumdaki bit özel bir ad ile adlandırılmıştır. Bu anlaşılabilirliği artırmak içindir. Programlarken yazmacın adı ise MCUCR olarak yazılmalıdır. Yazmacın adı ve bitlerin adları üretici tarafından belirlenmiştir ve programlarken teknik veri kitapçığına uyulmak zorunludur. Bu bitlere erişim ise yine üretici tarafından belirlenir. Bazıları sadece okunabilirken bazıları hem okunup hem yazılabilir. Y/O olarak belirtmemiz hem okunabilir hem yazılabilir olduğunu ifade etmek içindi. Şimdi bizi ilgilendiren PUD bitini açıklayalım.

PUD (Pull-Up Disable) Bu bit “1” yapıldığında tüm giriş ve çıkış portlarındaki dahili pull-up dirençleri devre dışı kalır. DDxn ve PORTxn yazmaçlarında pull-up tanımlansa dahi bu işlem gerçekleşir. Pull-Up dirençlerini devre dışı bırakmayı kaldırmak için bu bit tekrar sıfır yapılır.

MCUCLR yazmacının PUD bitini (4. bit) nasıl bir veya sıfır yapacağımızı ileride açıklayacağım. Şimdilik sadece yukarıdakileri okuyup anlamak yeterlidir. Şimdi giriş ve çıkış için kullanacağımız üç yazmacı anlatalım.

DDRx Bu yazmaç giriş ve çıkış portunun giriş mi çıkış mı olacağını belirler. Bir programı yazarken program başında hangi portun hangi ayağının hangi görevde olacağını bu yazmaca atadığımız değer ile belirleriz. Mikrodenetleyiciye göre kaç adet yazmaç olacağı değişiklik gösterir. Giriş ve çıkış yazmaçları alfabedeki harflerin sırasına göre adlandırılır. DDRA, DDRB, DDRC.. gibi her portun ayrı tanımlama yazmacı bulunur. DDRx (x burada portun harf adı) yazmacının genel yapısı şu şekildedir.

Yazmacın Biti	7	6	5	4	3	2	1	0
Karşılık Geldiği Ayak	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0

Yazmacın portun ayağına karşılık gelen biri bir (1) yapıldığında o ayak çıkış sıfır (0) yapıldığında o ayak giriş olur. Örneğin D portunun 3 numaralı ayağını giriş olarak tanımlamak istiyorsak DDRD’nin 3 numaralı (4. biti) bitini sıfır (0) yapmamız gerekir. İşlemcinin 8-bit olması burada 8-bitlik port ve yazmaçlar olmasına sebep olmuştur. Yine bu yazmaçlara değer atarken bayt olarak değer atayıp değer okurken ise bayt olarak değer okumamız gerekir.

PORTx

AVR mikrodenetleyicilerde giriş ve çıkış yapılan bitler 8-bitlik port olarak bir araya toplanmıştır. Her bir ayağı tamamen bağımsız olarak kontrol etmemiz yazılımsal olarak mümkünse de ayakların kendisi tamamen porttan yani ayak grubundan bağımsız değildir. Bu ayakların 8'li gruplara ayrılmasının en önemli nedenlerinden biri mikrodenetleyicinin 8-bit olmasıdır fakat daha önemli bir neden tek başına bağımsız bir ayaktan alınan giriş ve çıkışın fazla birşey ifade etmeyişidir. Tek bir düğmenin durumunu okuma, tek bir led yakma gibi basit işlemler için tek bir ayak yeterli olsa da 7-segman gösterge sürücüsüne bir değer göndermek istediğimizde 4 bitlik paralel bir hatta ihtiyacımız vardır. LCD kullanırken de 8 ya da 4 bitlik paralel bir hat gereklidir. İşte bu 8-bitlik portlar aynı zamanda 8-bitlik paralel iletişim portlarıdır. Bu portlar 0 ile 255 arasında değer alabilir ve bu değerin ikilik sistemdeki karşılığını ayaklara yansıtabilir. Aynı zamanda bu 0 ve 255 arasında yani bir bayt büyüklüğünde değeri de porttan okuyabilir. Pek çok dekoder ya da sürücü entegresi paralel bağlantıya ihtiyaç duyduğundan giriş ve çıkış portları gelişmiş uygulamalarda oldukça gereklidir.

Arduino programlayanların hiç alışık olmadığı hatta çoğunlukla adını bile duymadığı portlar AVR'nin temelini oluşturan birimlerden biridir. Arduino'da tek ayak üzerinden yapılan işlemler bile arka planda AVR portları üzerinden yapılmaktadır. Portların 8'li ayak grubu olması tek bir ayak üzerinden işlem yapamayacağımız anlamına gelmez. Fakat doğrudan değil dolaylı olarak bu işlemi gerçekleştiririz. Bunun hiçbir dezavantajı yoktur fakat yeni başlayanların biraz anlaması uzun sürmektedir. Örnek bir PORT yazmacının görünümü aşağıdaki gibidir.

Yazmacın Biti	7	6	5	4	3	2	1	0
Bulundurduğu Değer	0	0	1	0	1	1	1	1

Örnek yazmacımızın adı PORTD yazmacı olsun. PORTD'nin 0. bitinin değeri 1 olduğu için mikrodenetleyicinin PD0 ayağına taktığımız led yanacaktır. Bunu söndürmek için PORTD yazmacının 0. bitinin değerini 0 yapmamız gerekir. Bunu da C dilinin bitwise operatörleri ile yapıyoruz. Bunu daha sonra anlatacağız.

PINx

Bu yazmaç dijital giriş için kullanılır. Yani ayaklardaki elektriksel durumu okumayı sağlar. Bu bir düğmenin basılı olup olmadığını okumaktan 8-bitlik bir paralel iletişim bağlantısını okumaya kadar uzanabilir. DDRx yazmacı ile giriş olarak tanımlanan port ya da ayaklardan doğrudan port veya pin okuma yöntemi ile yazmaçtan elde edilen değer sonrasında mikrodenetleyicinin hafıza birimlerine kaydedilir ve bu değer üzerinde işlem yapılarak çıkış birimlerine iletilir. Burada verinin okunduktan sonra nasıl kaydedileceği, işleneceği ve çıkış olarak verileceği programcının yazdığı programa bağlıdır. Genel olarak mikrodenetleyici programlamak çevre birimlerinden değerleri okuyup bu değerleri kaydetmek, kaydedilen değerleri işledikten sonra kullanıcıya çıkış olarak vermek diye özetlenebilir. Bir algılayıcıdan sıcaklığı okuyup hesapladıktan sonra lcd ekrana yazdırmak ya da düğmeye basmakla ses seviyesini artırıp bu değeri de güncelleyerek ekrana yazdırmak gibi durumlar

buna örnektir. En temel değer okuma olan dijital giriş ise PINx yazmacı ile okunur. Yazmacı şekillendirerek anlatmak istersek şöyle bir tablo çizebiliriz,

Yazmacın Biti	7	6	5	4	3	2	1	0
Okunan Değer	0	0	1	0	1	1	0	0

Yazmacımız PIND yazmacı olsun. DDRD yazmacının 0. bitini 0 yaparak o ayağı giriş olarak tanımlıyoruz. Sonrasında pull-up direnci ile bir adet düğme bağladıktan sonra o düğmeye bastığımızda o ayağın değeri şaseye bağlandığından dolayı 0 oluyor. PIND ile okuma yaptığımızda 0. ayaktan tablodaki gibi 0 değeri elde ediyoruz. Düğmenin basılı olmadığı durumlarda ise 1 değeri okunacaktı.

Yazmaçlar anlaşıldıysa sonraki derste yazmaçlar üzerinde işlem yaparak giriş ve çıkış işlemlerinin nasıl olduğunu kodlarla örnekleyip anlatacağız.

Kaynaklar ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 17.08.2018

Temel Giriş ve Çıkış İşlemlerine Giriş

C ile AVR programlama derslerine bu derslere başlamadan çok önce başlamış ve sadece giriş ve çıkış işlemlerini anlatıp bırakmıştım. Önceki yazdığımı silip yedeği kaybettiğimden dolayı tekrar yazmak durumundayım. Bu sefer daha ayrıntılı ve açıklayıcı yazmaya gayret edeceğim.

Arduino'da temel giriş ve çıkış için üç adet fonksiyon kullanıyorduk. Bu üç fonksiyonun kullanımı oldukça kolay ve anlaşılırdı. AVR'de ise yeni başlayanların kafasını karıştıracak şekiller ve kısaltmalar karşımıza çıkıyor. Bu kısaltmaları yani yazmaç adlarını önceki yazıda uzunca anlatıp iyi bir hazırlık yaptık. Şimdi ise bu şekilleri yani operatörleri ve bunlar üzerinden nasıl işlem yapıldığını anlatacağız.

Arduino'da tek bir ayaktan çıkış almak için digitalWrite() tek bir ayağı okumak için digitalRead() ve ayağın görevini tanımlamak için pinMode() fonksiyonları kullanılır. Üç görev için üç fonksiyon kullanılmaktadır. Bu durum AVR'de de çok farklı değildir. Yine üç görev için üç yazmaç kullanacağız ve fonksiyonlar aracılığı ile yazmaçlara değer atamak yerine doğrudan yazmaçlara müdahale edeceğiz. Böylelikle programımız çok daha hızlı olacak ve az yer kaplayacak.

AVR mikrodenetleyicilerde portların ayakları Input, Input Pull-up, Sink ve Source konumlarında olabilir. Input konumunda harici olarak pull-up veya pull-down yani direnç ile beslemeye veya şaseye bağlanmış halde ya da dahili pull-up direncine bağlı halde olur. Output konumunda ise Sink ve Source olarak iki durumda olur. Sink; 0, LOW, FALSE

konumu olup 0V olup 20mA civarında akım çeker. O yüzden sink yani akmak olarak adlandırılır. Source ise 5v 20mA civarında akım verir ve mantıksal olarak HIGH, TRUE ya da 1 olarak adlandırılır. Source'un kelime anlamı ise kaynaktır. Bundan başka Tri-state diye adlandırılan bir durum vardır. Bu üçüncü durum olup ne mantıksal HIGH ne de mantıksal LOW demektir. Hükmü olmayan bir durumu tri-state olarak adlandırırız.

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

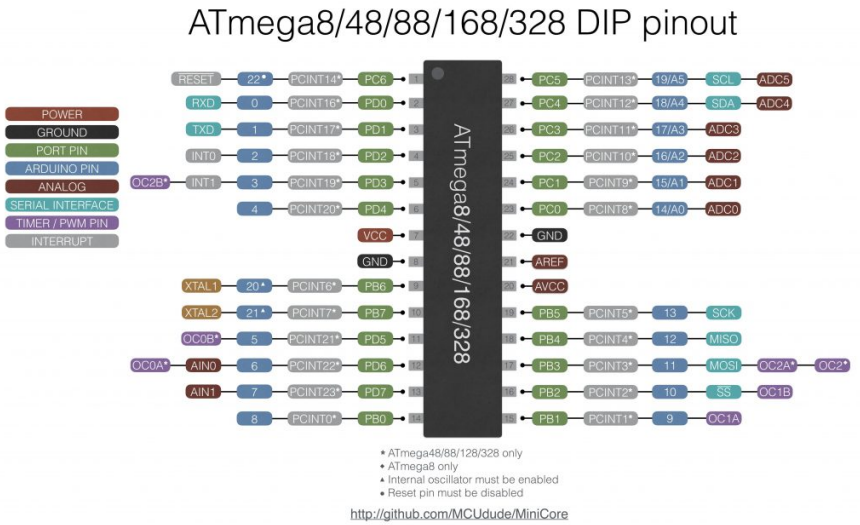
Resim: ATmega328P – Microchip Technology, sf.99 , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 17.08.2018

Yukarı teknik veri kitapçığından alınan resimde AVR mikrodenetleyicilerin pin konfigürasyonu verilmiştir. Burada MCUCR yazmacındaki PUD bitinin giriş olarak tanımlanan port ve bitlerindeki görevini görmemiz mümkün. Tabloyu madde madde şöyle özetleyebiliriz,

- DDRx yazmacındaki porta ait ayak sıfır yapılır ve port yazmacındaki aynı bit de sıfır yapılırsa giriş ve çıkış fonksiyonu giriş olur ve tri-state olarak kalır. (Ayak boşta iken bu durum geçerlidir.)
- DDRx yazmacındaki porta ait ayak sıfır yapılır ve port yazmacındaki aynı bit de bir yapılırsa ve PUD biti sıfır konumunda yani pull-up dirençleri devre dışı bırakılmamışsa o ayak dahili pull-up özelliği gösterir.
- DDRx yazmacındaki porta ait ayak sıfır yapılır ve port yazmacındaki aynı ayağa karşılık gelen bit de bir yapılır ama MCUCR yazmacındaki PUD biti 1 yapılırsa o ayak tri-state konumunda olur. Yani pull-up olmadan giriş konumunda olur.
- DDRx yazmacındaki porta ait ayak bir yapılır ve port yazmacındaki aynı ayağa karşılık gelen bit de sıfır yapılırsa o ayaktan dijital 0 (sıfır) çıkışı sağlanır.
- DDRx yazmacındaki porta ait ayak bir yapılır ve port yazmacındaki aynı ayağa karşılık gelen bit de bir yapılırsa o ayaktan dijital 1 (bir) çıkışı sağlanır.

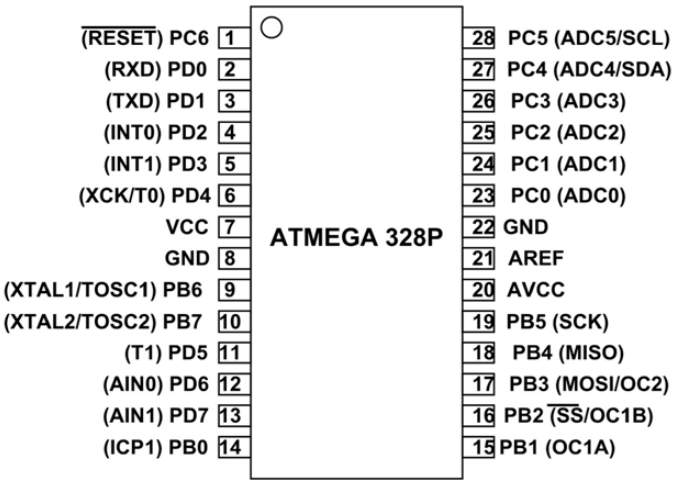
Bu tablo aslında bizim önceki yazıda anlattığımız yazmaçlardaki bitlerin konumuna göre ayaklardan nasıl bir giriş ve çıkışı alacağımızı anlatıyor. Eğer önceki başlığı okuyup geldiyseniz burayı anlamak hiç zor olmayacaktır. Şimdilik MCUCR yazmacındaki PUD bitini bir kenara bırakalım ve sadece anlattığımız üç yazmaçtaki bitlerin konumuna göre ayakları nasıl kontrol edeceğimizi anlatalım fakat bundan öncesinde Atmega328p

mikrodenetleyicisinin ayaklarını öğrenmekle başlayalım.



Resim : http://www.ifuturetech.org/ifuture/uploads/2013/07/ATmega328P-PU-PIN-Diagram-connection-configration.jpg (*Resme tıklayarak tam çözünürlükte açabilirsiniz.*)

Yukarıdaki şemadan anlaşılacağı gibi mikrodenetleyicinin ayakları portlar ve diğer birimler ile ortak kullanılmaktadır. Bir ayağı hem port, hem iletişim, hem analog hem de kesme olarak kullanmak mümkündür. Fakat biz burada diğer birimleri şimdilik anlatmayacağız ve portları anlatmakla işe başlayacağız. Atmega328P mikrodenetleyicisinin güç ayakları hariç geri kalan tüm ayaklarını portlar oluşturur. Bu portlar PORTB, PORTC ve PORTD olarak üç adettir. PORTD tam 8-bite karşılık gelen 8 ayaktan oluşsa da diğer portlar 8 ayaktan oluşmamaktadır. Mikrodenetleyicide ayaklar kısıtlı olduğu için bu 8-bit çıkış alabileceğimiz tek bir port bulunur. Üstelik UART, I2c, Analog ve SPI kısımları bu portlarla paylaşıldığı için gelişmiş uygulamalarda portlar için kullanılacak ayaklar yetersiz gelebilir. Şimdilik her özelliği kullanmayacağımız için elimizde oldukça fazla ayak var demektir. Fakat Arduino UNO kartında UART kullanıldığı için RX ve TX ayaklarına denk gelen PD0 ve PD1 ayaklarını kullanmamamız iyi olur. Burada ayak adlandırmaları Pxn şeklinde olup örneğin PD1 , PORTD'in 1 numaralı ayağı anlamına gelmektedir. Yukarıdaki resmi kullanışlı bulmayanlar için aşağıda pratik bir referans resmini verelim.



Resim: https://components101.com/sites/default/files/component_pin/ATMega328P-Pinout.png

Led Yakma

Şimdi basit bir led yakma programı yazalım ve bu program üzerinden yazmaçların nasıl programlandığını anlatalım. Arduino UNO kartı kullandığımız için şimdilik devre kurmamıza gerek yok. Programı karta atalım ve çalıştıralım. Programı yüklediğimizde L isimli PB5'e bağlı led yanacaktır.

```
#include <avr/io.h>

int main(void)
{
        DDRB = 0xFF; // 0b11111111
        PORTB = 0xFF; //0b11111111
    while(1)
    {

    }
}
```

#include <avr/io.h> direktifi ile temel giriş çıkış başlık dosyasını programımıza ekliyoruz. C dilinde stdio.h başlık dosyası eklendiği gibi giriş çıkış ve port işlemleri için io.h başlık dosyası programın başında eklenmelidir.

int main(void) fonksiyonu ana program fonksiyonudur. Ana programı main içerisine yazmak zorundayız. Bu fonksiyon içine yazdığımız komutlar yukarıdan aşağıya doğru bir defalığına işletilir.

DDRB = 0xFF; ifadesi DDRB yazmacına 0xFF yani ikilik tabanda "11111111" değerini aktarır. Mümkün olduğu kadar değerleri onaltılık tabanda yazmamız bizim için daha pratik olacaktır. Anlaşılır olması için şimdilik onaltılık tabandaki sayıların ikilik tabandaki karşılıklarını veriyoruz. DDRB yazmacına "11111111" değeri atılınca D portunun bütün ayakları çıkış olarak ayarlanmış olur. Böylelikle bu ayaklardan HIGH (1) ya da LOW (0) çıkışı alınabilir.

PORTB = 0xFF; Bu ifade ile PORTB yazmacına 0xFF yani ikilik tabanda "11111111" değerini aktarıyoruz. Böylelikle tüm ayaklardan HIGH (1) çıkış alınmış oluyor. Böyle çıkış alındığı zaman bu ayaklara bağlı olan led ya da buzzer gibi elemanlara elektrik gitmiş oluyor ve bunları çalıştırıyor. Eğer sıfır değerini atasaydık bağlı olduğu led sönecekti. C dilinde onaltılık değerler 0x öneki ile ikilik değerler ise 0b öneki ile ifade edilir. Onluk tabandaki sayıları düz yazmak yeterlidir fakat bu tabanlardaki değerleri ifade etmek için değerin önüne bu önekleri (prefix) koymak gereklidir.

while(1) kısmı programın sonsuz döngüsüdür. Arduino'dan farklı olarak main fonksiyonu sonsuz döngü şeklinde değil bir defaya mahsus çalışır. O yüzden sonsuz program döngüsüne ihtiyacımız olduğu durumda bu ifadeyi kullanırız. Bu ifadenin arasına yazılan kodlar çalışma boyunca sürekli işletilir.

Görüldüğü gibi önce DDRx yazmacına bir değer atıyoruz ve sonra atadığımız değere göre portların ayakları çıkış olarak ayarlanıyor. Çıkış olarak ayarlandıktan sonra PORTx yazmacına atadığımız değerlere göre ayaklar bir (1) veya sıfır (0) oluyor.

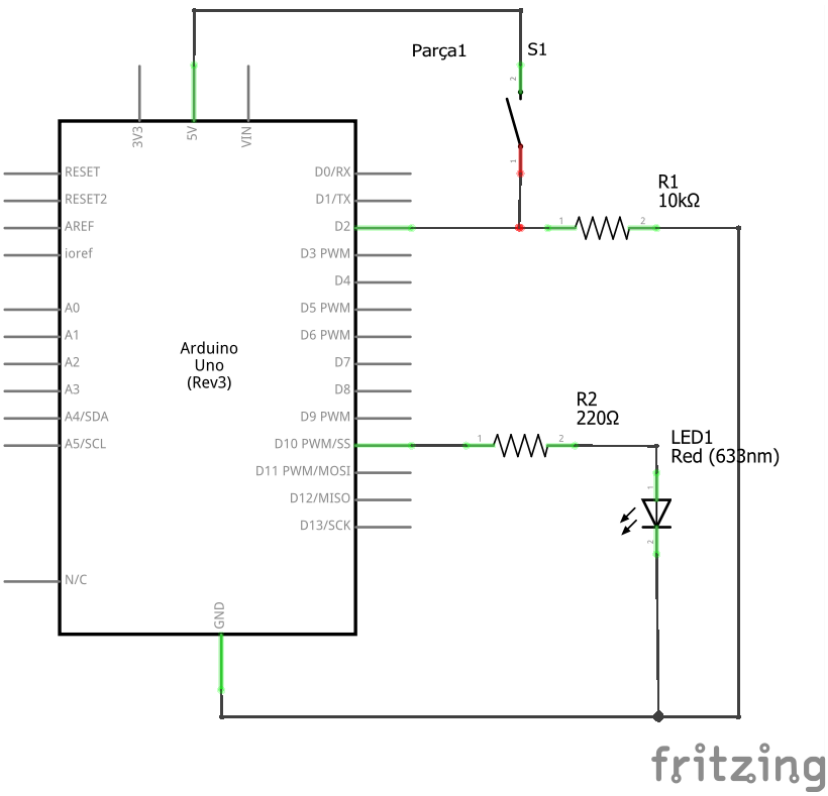
Sonraki konularda port yazmaçları üzerinde yapılan işlemleri en ince ayrıntısına kadar anlatacağız. Anlaşılması için konunun uzun tutulması gerektiğinden bu dersi burada bitirelim.

Kaynaklar ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 17.08.2018

Port Üzerinden Giriş/Çıkış ve Delay Kütüphanesi

AVR programlamada portlar ve bunlar üzerinde yapılan işlemlerden devam ediyoruz. Bu konuyu bu kadar uzatıp birkaç derse yaymamın sebebi ise AVR'ye geçiş için en büyük adım olmasından dolayıdır. Bu konu anlaşıldığı zaman diğer konular çok kolay anlaşılacaktır. Gömülü sistemlerde C dilini kullanmayı, yazmaçları ve mikrodenetleyicinin mantığını anlamak için gerektiği kadar uygulama yapıp başlangıçta ağır ağır ilerlememiz lazımdır. Bir örnek kod verip iki satır açıklama yaparak bu konuları anlatmak mümkün değildir.

Önceki yazımızda port yazmaçlarına değer atamış ve bu değere göre bir ledi yakmıştık. Şimdi DDRx, PORTx ve PINx yazmaçlarını bir arada kullanacağımız bir uygulama yapacağız. Böylelikle üç yazmacın tüm özelliklerini uygulamada görmüş olacağız. Sonraki derste ise bitwise operatörleri ve makrolar ile yazmaçlar üzerinde bit tabanlı işlem yapacağız. Makrolar işimizi kolaylaştırsa da işin mantığını anlamamıza engel olmaktadır. O yüzden zordan başlayıp derslerin sonunda da makroları kullanacağız. Şimdilik şöyle bir devre kuralım ve bunun üzerinden bu üç yazmacı anlatalım.



Arduino UNO'da Dijital 2. ayak Atmega328P'de PD2'ye 10. ayak ise PB2'ye denk gelmektedir. Kısa sürede hatırlamak için kağıda kabaca bir Arduino resmi çizip ayakların yanına Atmega328P'nin ayak adlarını yazabilirsiniz. Oldukça faydası oluyor. Şimdilik devreyi Arduino üzerinden verelim fakat ileride entegre üzerinden vereceğiz ve Arduino kartında hangi ayağa denk geldiğini bulmak sizin sorumluluğunuzda olacak. Bu devrede pull-up ve pull-down yöntemini kullandık. Bunu açıklamadan önce örnek pull-up ve pull-down devrelerini resimde verelim.

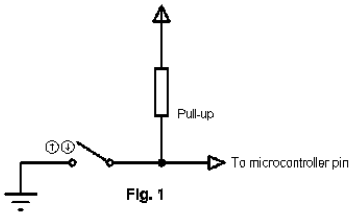


Fig. 1

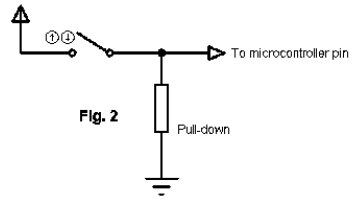


Fig. 2

Resim : http://images.elektroda.net/37_1290066934.png

Dijital devrelerde ayağın açıkta olması tri-state adı verilen kararsız bir durum ortaya çıkarır. Ne bir (1) ne de sıfır (0) olan ve bir şey bağlanmadığı sürece okunduğunda bir anlam ifade etmeyen kararsız bir durumdur. Buna mantık kararsızlığı ya da lojik kararsızlık da denebilir. Bunu önlemek için ayağı boşta bırakmak yerine bir direnç ile ya beslemeye ya da şaseye bağlayıp bir (1) veya sıfır (0) yapmamız gerekir. Böylelikle o ayağın sabit değerini belirlemiş oluruz ve farklı bir değeri bu sabit değere göre saptayabiliriz. Örneğin pull-up direnci ile beslemeye bağladığımız bir giriş ayağı şaseye de düğme ile bağlanmış olsun. Bu ayaktan düğmeye basılı olmadığı zaman alacağımız değer sabittir ve her zaman bir (1) olarak alınır. Düğmeye bastığımızda ayak şase ile kısa devre olur ve sıfır konumuna düşer. Bu durumda ayaktan sıfır (0) değerini okuruz ve muhakkak düğmeye basıldığını anlarız. Yoksa elektriksel gürültüye göre bir veya sıfır değerlerini rastgele almamız mümkündür. Onun için dijital giriş devrelerinde pull-up ve pull-down devrelerini ihmal etmemek gerekir. Bu dirençlerin değeri 10K olabilir.

Devreye dönersek pull-down olarak bağlandığı için o ayaktan sürekli sıfır verisi okunması gerekir. Düğme beslemeye bağlandığı için düğmeye basıldığı anda bir (1) değeri okunacaktır.

PB2 ayağına ise bir led 220ohm direnç ile bağlanmıştır. Artık düğmeye bastığımız anda ledi yakacak bir programa ihtiyacımız vardır. O program ise aşağıdaki gibidir.

```
#include <avr/io.h>

int main(void)
{
        DDRD = 0x00;
        DDRB = 0xFF;
    while(1)
    {
    PORTB = PIND;
    }
}
```

DDRD = 0x00; Bu komutla D portunun tüm ayakları giriş olarak tanımlanır. Burada bit veya ayak dememizin bir önemi yoktur ikisi de aynı anlama gelmektedir.

DDRB = 0xFF; Bu komutla B portunun tüm ayakları çıkış olarak tanımlanır. Giriş veya çıkış olarak tanımlamak bizim atadığımız değere göredir. Bir kısmını giriş veya bir kısmını çıkış yapmamız da yine atadığımız değere göre olmaktadır. Örneğin 0b11110000 değerini atasaydık ilk 4 bacak giriş son 4 bacak çıkış olarak tanımlanacaktı. Bunun onaltılık karşılığı ise 0xF0'dır.

PORTB = PIND; İşte bütün işi yapan ve en önemli kodumuz bu koddur. Burada önceki programdan farklı olarak PORT yazmacına bir sabit değer atamak ve buna göre ayaklardan çıkış almak yerine PIN yazmacının değerini doğrudan PORT yazmacına attık. Bu kodun Türkçesi şudur, PORTD'ye bağlı ayaklarda okunan değeri PORTB yazmacına aktar. PIN yazmacı giriş olarak tanımlanan port ayaklarının ayak durumunun verisini içerisinde barındırır. Bu veri bir değişkene atanıp kullanılabilir. Örneğin 8-bitlik bir paralel iletişim hattı kurduk ve bir portun 8 ayağı da karşı cihaza bağlı. Bu karşı cihazdan aldığımız 8-bitlik veriyi degisken = PINx; komutu ile bir değişkene atayabiliriz. 8-bit mikrodenetleyici olduğu için DDR, PORT, PIN yazmaçlarının hepsi 0 ile 255 değer aralığında olmaktadır.

Özetlemek gerekirse, bu devre düğmeye basınca ledi yakmaktadır düğme basılı değilken ise led sönmektedir. Ledin bu durumu düğmeye bağlı ayağın durumuna eşittir.

Bekleme işlemi ve Delay kütüphanesi

Mikrodenetleyicilerde delay yani bekleme komutu sıkça kullanılır. Mikrodenetleyici bilgisayarlara göre çok daha düşük hızda çalışsalar da yazılı komutları peş peşe çalıştırdıkları için günlük hayata göre oldukça hızlı çalışmaktadırlar. Çoğu zaman komutlar arasına biraz bekleme süresi koymamız gerekebilir. Komutların ne kadar hızlı çalıştığı değil zamanında çalışması daha önemlidir. Bekleme özelliği aslında donanımsal bir özellik olmayıp yazılım ile mikrodenetleyiciyi boş yere meşgul etmekten ibarettir. Biz delay kütüphanesini kullansak da internette örnek delay komutları bulmanız mümkündür. Ana mantık mikrodenetleyicinin hızlı işlem gücünü boş bir yerde harcatıp mikrodenetleyiciyi bir süre meşgul etmektir. Zamana bağlı uygulamalarda delay komutu asla ilk sırada olmamalı ve zamanlayıcılar büyük ölçüde kullanılmalıdır. Aşırı derecede kullanılan delay komutu mikrodenetleyiciyi olduğundan fazla yavaşlatır ve gerekli komutların zamanında çalışmasına engel olur. Şimdilik delay kullanmamızda bir sakınca olmadığı için yanıp sönen led uygulamasıyla delay komutlarını anlatalım.

```
#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>

int main(void)
{
        DDRB = 0xFF;
    while(1)
    {
    PORTB = 0xFF;
        _delay_ms(1000);
        PORTB = 0x00;
        _delay_ms(1000);
    }
}
```

Bu program B portuna bağlı tüm ayakları yakıp söndürecektir. B portuna bağlı dahili bir ledimiz bulunduğu için herhangi bir devre kurmamıza gerek yoktur. Arduino kartının üzerindeki ledin birer saniye aralıklarla yanıp söndüğünü göreceksiniz. Şimdi kodu bütün ayrıntısıyla inceleyelim ve sonrasında delay kütüphanesinin tamamını anlatalım.

#define F_CPU 16000000 F_CPU değerinin 16000000 olduğunu belirledik. Bu delay kütüphanesinin kullanacağı bir değer olup mikrodenetleyicinin saat hızını öğrenmek için kullanılır. Mikrodenetleyici kaç MHz'de çalışıyorsa onun Hertz değeri yazılmalıdır. Bizdeki 16MHz olduğu için delay.h kütüphanesini dahil etmeden önce o değeri belirledik.

#include <util/delay.h> Delay kütüphanesini çağırıyoruz. Burada delay kütüphanesine ait olan fonkisiyon ve değerler programda kullanılabilir oluyor.

DDRB = 0xFF; B Portunun bütün ayaklarını çıkış olarak tanımlıyoruz.

PORTB = 0xFF; B portunun bütün ayaklarını bir (1) konumuna getiriyoruz böylelikle lambayı yakıyoruz.

_delay_ms(1000); 1000 milisaniye bekliyoruz. _delay_ms() fonksiyonunun içindeki paranteze yazdığımız sabit değer veya değişken kadar bekler.

PORTB = 0x00; B portunun bütün ayaklarını sıfır (0) konumuna getiriyoruz böylelikle lambamız sönüyor.

_delay_us()

Bu fonksiyon ise mikrosaniye kadar bekletmeyi sağlar. Parantezin içerisine double değer aralığında değer yazılabilir. Diğer fonksiyon için de bu geçerlidir. F_CPU ise unsigned long olabilir.

Tekrar özetlersek, DDRx, Bir portun giriş mi çıkış mı olacağını belirler PORTx, Bir portun ayaklarının bir (1) mi sıfır (0) mı olacağını belirler. PINx, Bir porttaki girişlerin değerini okur.

Buraya kadar portlar üzerinde kabaca işlem yaptık ve bitlerine doğrudan müdahale etmedik. Bundan sonraki yazıda bitwise operatörler ile bunlara müdahale edeceğiz. Sonrasında ise makroları kullanmayı anlatacağız. Böylelikle dijital giriş ve çıkış konusunda anlatılmadık bir şey kalmayacak.

Kaynaklar: <util/delay.h>: Convenience functions for busy-wait delay loops, https://www.nongnu.org/avr-libc/user-manual/group__util__delay.html, Erişim Tarihi : 17.08.2018

ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 17.08.2018

Bit Tabanlı Giriş ve Çıkış

Önceki derste görüldüğü üzere portlara değer atamakla giriş ve çıkış işlemini yapıyorduk. Bir port yazmacına değer atayarak giriş ve çıkışı yapmak ancak bir portu bütünüyle kullanırken kullanışlı olabilir. Aksi halde sadece değer atayarak giriş ve çıkış kontrolünü sağlamak oldukça yetersiz kalır. Bunun için bitwise operatörler ve makroları kullanmaktayız. C dilinde bulunan bitwise operatörler bir değer üzerinde bit ve mantık işlemlerini yapmayı sağlar. Böylelikle bitlerden oluşan port yazmacına bit seviyesinde müdahale edebiliriz.

Değer Atama Yoluyla Port Denetimi

Bu yol en basit yol olup “=” operatörü ile port yazmacına istediğimiz sabit bir değeri veya değişkeni atayabiliriz. 8-bitlik portun bütün bitleri atadığımız değere göre güncellenir. O

yüzden ayrı ayrı değil bütünüyle portu ele almamız gerekir. Bu değer atama şu şekilde olabilir,

```
PORTB = 0xFF; // PortB'nin tüm ayakları 1 (HIGH) olarak ayarlandı.
PORTD = 0x00; // PORTD'nin tüm ayakları 0 (LOW) olarak ayarlandı.
PORTE = 0b11110000; // E portunun ilk 4 ayağı 0 (LOW) son 4 ayağı 1 (HIGH)
olarak ayarlandı.
```

Kayan ışık programı yapmak istediğimizde ise portlara her defasında sıfırdan değer atamamız gerekecektir. Yani sekiz bite de baştan değer atamak gerekecektir.

```
PORTE = 0b00000001;
PORTE = 0b00000010;
PORTE = 0b00000100;
PORTE = 0b00001000;
PORTE = 0b00010000;
PORTE = 0b00100000;
PORTE = 0b01000000;
PORTE = 0b10000000;
```

Eğer istediğimiz bir led yanıyor ve öteki ledi de yakmak istiyorsak hem önceki ledi yakmak ve hem sonraki ledi yakmak durumunda kalacağız. Eğer bir ledi yakmak isterken diğer atadığımız değerler öncekinden farklı olursa istenmeyen bir değişiklik meydana gelecektir. Bunun için diğer bitleri önemsemeyecek, bitleri kaydıracak, bitleri değiştirecek veya bitlere istenmedikçe dokunmayacak bir özellik gereklidir. Bunlar bitwise operatörleri ile sağlanır.

Bitwise Operatörleri ile Port Manipülasyonu

Bitwise Operatörler C ile gömülü sistem programcılığında olmazsa olmazlardandır. C dilinin alt seviye özelliklere sahip bir programlama dili olduğunu söylemeliyiz. Bu alt seviye özelliklerin başında ise işaretçiler (pointer) ve bitwise operatörler gelir. Bu alt seviye özellikler sayesinde daha alt seviye bir dile muhtaç olmadan hem de alt seviye dilde olmayan pek çok özelliğe sahip olarak pek çok gelişmiş program yazılabilir. Günümüzde gömülü sistemlerde C dili bir numaralı dil olarak canlılığını hiç kaybetmeden kullanılmaktadır. Şimdi bitwise operatörleri sayıp sonrasında örneklerle açıklayalım.

- & : AND Operatörü
- | : OR Operatörü
- ^ : XOR Operatörü
- ~ : Tersleme Operatörü
- << : Sola kaydırma Operatörü
- >> : Sağa Kaydırma Operatörü

Bu operatörler C'deki mantıksal operatörlerle karıştırılmamalıdır. Herhangi bir mantıksal karşılaştırma yapmayan bu operatörler sadece bitler üzerinde işlem yapar.

Bir porttaki bir ayağı bir (1) konumuna getirmek istiyoruz diyelim. Burada porttaki diğer ayakların durumunu bilmiyoruz ve bu ayaklara dokunulmadan sadece bizim belirttiğimiz bitte değişiklik yapılacak. Bunun için portun değeri ile bizim atadığımız değere OR işlemi yaptırıp sonra tekrar porta atamamız gerekir. D portunun son birine bağlı bir ledi yakmak istiyorsak bunun için şöyle bir işlem yapmalıyız,

PORTD = PORTD | 00000001;

Bu işlemi tabloda anlatmamız gerekirse. Tablodaki ilk satır D portunun ilk hali, ortadaki satır bizim atadığımız değer en son satır ise D portunun son halidir.

0	1	0	1	1	1	0	0
0	0	0	0	0	0	0	1
0	1	0	1	1	1	0	1

Görüldüğü gibi D portunun diğer bitlerine herhangi bir müdahalede bulunulmayıp ilk bitine 1 biti ile OR işlemi uygulanmıştır. Bu işlemin doğruluk tablosuna göre sonuç 1 olması gerektiği için o bit 1 ile değiştirilmiştir. Eğer bit 1 olsaydı yine bir olacaktı. Sıfır yazdığımız yerlerde de sıfır veya bir olmasına bakılmadan aynı değer OR işlemi ile elde edilmiştir. Bu işlemde 1 yapmak istediğimiz yere 1 dokunmak istemediğimiz yere 0 yazıyoruz. Böylelikle atama operatöründeki gibi değerler tamamen değişmiyor. Arduino'da olduğu gibi tek bir bacağı HIGH ya da LOW yapmak için bir komut C dilinde yoktur. Mikrodenetleyicinin donanımında ise giriş ve çıkış tek bitlerden değil portlardan oluşmaktadır. O yüzden böyle bir çözüm üretiyoruz.

Şimdi bu kodu bu kadar uzun yazmak yerine kısaltarak yazalım. C dilinde aritmetik operatörlerde nasıl kısaltma söz konusu oluyorsa bu bitwise operatörlerde de kısaltma söz konusudur. Yukarıda koyu renkte yazdığımız kodun daha kısa bir ifadesi şu şekildedir.

PORTD |= 00000001;

Şimdi bu işaretin neye karşılık geldiğini önce uzun yolu öğrenerek rahatça anlayabilirsiniz. Bu işareti başta anlatsaydık büyük ihtimalle ezberleyecek ve asıl işlevini anlamadan kullanacaktınız. Bu işleme terim olarak "maskeleme" de denmektedir. Şimdi ise sağ taraftaki değeri kısaltarak işe devam edelim. AVR mikrodenetleyicilerde her portun bir ayağının adı vardır. Bu ayağın adını zikrederek bu ayak üzerinde işlem yapabiliriz. Örneğin D portunun 5. ayağını PORTD5 olarak zikredebiliriz. Şimdi D portunun 0. ayağı üzerinden aynı kodu tekrar yazalım.

PORTD |= (1<<PORTD0);

Bu komut yukarıdaki komuttan daha pratik ve daha anlaşılır oldu. Ama sadece bununla sınırlı kalmıyoruz ve kısaltmaya devam ediyoruz. Bu noktadan sonra hangi şekilde kullanılacağı programcının tercihine kalmış bir durumdur.

PORTD |= (1<<PD0);

Yukarıdaki kadar olmasa da yine de P ve D harflerinden anlaşılır bir komut oldu. PORTD0 ile PD0 aynı işlevdedir. Şimdi ise benim tercih ettiğim yöntem olan en kısa yöntemi anlatıp bu "<<" operatörünün aslında ne işe yaradığını açıklayacağız.

PORTD |= (1<<0);

Burada işaretler yığını görünen bir komutta aslında çok basit bir işlem yapılmaktadır. En sağda ayak numarası (aslında bitin kaç adım sola kaydırılacağı) << işaretinin solunda bitin kendisi ortadaki |= ise maskeleme için kullanılan OR işleminin kısaltılmışı olarak yer alıyor. Burada sola kaydırma (<<) operatörünü aslında yukarıda 0 ve 1'ler ile yazdığımız değeri elde etmek için kullanılıyoruz. Örneğin (1<<1) yazdığımızda bu "00000010" sonucunu veriyor. Yani "1" değerindeki biti sola bir kere kaydır demiş oluyoruz. Aynı şekilde (1<<7) dediğimizde "10000000" sonucunu elde ediyoruz. Yani "1" değerindeki biti sola 7 kere kaydırmış oluyoruz.

Yukarıda verdiğim en basit örnek ile en karmaşık örnek arasında değer bakımından değil kısaltma ve pratikleştirme açısından fark vardır. Her durumda aynı işlemi yapmış oluyoruz. İnternette incelediğim 10'a yakın İngilizce ders yazılarının hiçbirinde bunlar bu kadar ayrıntılı ve toplu halde açıklanmamıştır. Bunları bir yönden anlatıp geçmek de mümkündür ama biz bu kısa kesmek istemiyoruz.

Şimdi **HIGH** yapmak için kullanılan komutları sözdizimsel şekilde tekrar toplayalım.

PORTx |= (1<<PORTxn); PORTx |= (1<<Pxn); PORTx |= (1<<n);

x: port adı n: ayak numarası

Buraya kadar bir biti bir (1) yani HIGH yapmayı anlattık. Şimdi ise bir biti (0) yani LOW yapmaya sıra geldi. Fakat burada OR operatörü ile bir biti sıfır yapmamız mümkün değil. O halde farklı bir işleme ihtiyacımız var. Yine aynı mantıkta fakat bu sefer aradığımız işlemi yapıp uygun sonucu verecek bir komuta ihtiyacımız var. Bunun için AND ve tersleme (~) operatörlerini kullanacağız. Örneğin D portunun 0. bitini HIGH yaparak ona bağlı olan ledi yaktık. Yanıp sönen led uygulaması yaptığımızdan dolayı belli bir süre sonra söndürmek istedik. Bu durumda şöyle bir kod yazacağız.

PORTD = PORTD & ~00000001;

Öncelikle tersleme operatörü belirlediğimiz sabit değeri tersleyerek işleme başlayacaktır. Değer terslendikten sonra "11111110" olacaktır. Bu değer ile PORTD'nin değeri AND işlemine tabi tutulup elde edilen sonuç PORTD'ye aktarılacaktır. Bunu yine tablo ile verelim ilk değerimiz PORTD'nin ilk değeri olsun, ikinci değerimiz bizim belirlediğimiz sabit değer ve son değer ise PORTD'nin güncellenen değeri olsun.

0	1	0	1	1	1	0	1
1	1	1	1	1	1	1	0
0	1	0	1	1	1	0	0

Görüldüğü gibi 1 olan yerlere 1 ile AND işlemi yapıldığında 1 oluyor fakat 0 olan yerlere 0 ile AND işlemi yapıldığında 0 oluyor. Yani diğer hiçbir bitin işleme tabi tutulsa bile

değerinde değişme olmuyor. Fakat bizim 0 ile AND işlemine tabi tuttuğumuz bit 1 durumunda ise 0 oluyor. Yani bu durumda HIGH'dan LOW seviyesine çekmiş oluyoruz. Eğer 0 olsa idi yine bir değişme olmayacaktı. Her durumda istenilen bitte LOW sonucunu elde ediyoruz.

Buraya kadar anlaşıldıysa artık bunu da kısaltarak yazıp kullanabiliriz. Öncelikle operatör tarafından bir kısaltma yaparak AND operatörünü kısaltalım.

PORTD &= ~00000001;

Yine yukarıdaki gibi bir kısaltma yaparak kodumuzu daha anlaşılır bir seviyeye çekebiliriz.

PORTD &= ~(1<<PORTD0);

Şimdi yine PORT kısmından kısaltalım ve PD0 olarak yazalım

PORTD &= ~(1<<PD0);

En sonunda ise en kısa sürümüne geçelim. En anlaşılmaz gibi gözüken bu sürümün aslında en saf ve en anlaşılır olanı olduğunu söylemek yanlış olmaz.

PORTD &= ~(1<<0);

Şimdi **LOW** yapmak için kullanılan komutları sözdizimsel şekilde tekrar toplayalım.

PORTx &= ~(1<<PORTxn); PORTx &= ~ (1<<Pxn); PORTx &= ~ (1<<n);

x: port adı n: ayak numarası

Buraya kadar dijital çıkışta öğrenmeniz gereken bilgiyi fazlasıyla anlattık. Bir ledi yakıp söndürmek en uzun böyle anlatılabilirdi diyebilirsiniz fakat anlamamıza gerek olmasa da isteyenler için makrolarla kullanımı da sonraki yazıda anlatacağız. Makrolarla dijital giriş birbiriyle alakalı olduğundan ikisini beraber anlatmak üzere dijital girişi sonraki yazıya bırakıyorum.

Kaynaklar:

Bitwise Operators in C Programming, https://www.programiz.com/c-programming/bitwise-operators, Erişim Tarihi: 18.08.2018

ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 18.08.2018

Dijital Giriş ve Makrolar ile Dijital Giriş ve Çıkış

Önceki yazımızda operatörler ile bit tabanlı dijital çıkışı anlatmış ve dijital giriş ile makroları bu yazıya bırakmıştık. Artık bu yazıda dijital giriş ve çıkış ile alakalı tüm konuları anlatmış olacağız ve diğer konulara geçeceğiz. Fakat bu konuyu burada bırakmayıp ileride örnek devre ve program üzerinden anlatmaya devam edeceğiz. Makrolar operatörlerle yapılan işleri biraz daha anlaşılabilir kılıp kolaylaştırmak için kullanılan fonksiyonlardır. Bu fonksiyonları hiç kullanmadan da bütün işleri yapmamız mümkündür. Fakat programın okunabilirliği açısından makroların inkar edilemez bir faydası vardır. Burada makro kullanmayı sizin tercihinize bıraksam da dijital çıkışta makro kullanmayıp dijital girişte makro kullanmayı tavsiye ediyorum. Dijital girişte makrosuz bir programlama zaman zaman karışıklıklara sebep olabilir.

Makrosuz Bit Tabanlı Dijital Giriş

Dijital girişte PINx yazmacındaki değeri kullanıyorduk. PINx yazmacındaki değer 8-bitlik olup tek bir bite erişimi değer atama (=) yöntemiyle sağlayamıyorduk. Aynı dijital çıkışta olduğu gibi bunda da bitwise operatörler ile tek bir bite erişim sağlayabiliriz. Bunun için **PINx & (1<<PORTxn);** komutunu kullanıyoruz. Örneğin D portunun 1 numaralı bacağından bir dijital okuma yapıp değişkene aktaralım.

```
degisken = PIND & (1<<PORTD1);
```

Burada okuyacağımız değer HIGH ve LOW çeşidinden olacağı için sıfırdan farklı bir değer HIGH olarak okunacaktır. Genellikle if gibi karar yapılarında düğme durumu okunduğu için okunan değer değil değerin sıfırdan farklı olup olmaması önemlidir. Bunu kısaltarak yazmak istersek yine aşağıdaki iki şekilde yazabiliriz.

```
degisken = PIND & (1<<PD1);
```

```
degisken = PIND & (1<<1);
```

Son komut üzerinden bu işlemin nasıl çalıştığını anlatalım. Öncelikle “1” değerindeki ikilik sayı “<<” operatörü ile bir bit kadar sola kaydırılır. Böylelikle 1 numaralı bite denk gelen değer yani “00000010” elde edilmiş olur. PIND ise giriş ayaklarının ayak durumunu barındıran yazmaçtır. PIND’deki orjinal değer ile AND işlemine tutulan bizim değerimiz eğer ayak gerçekten “1” konumunda ise sonuç kısmındaki bite “1” değerini atar. Eğer ayak sıfır konumunda ise AND işlemine tabi tutulduğundan o değer “0” olur. Bu durumda belirtilen

ayağın durumu öğrenilmiş ve sonuca aktarılmış olur. D portunun birinci ayağı “1” konumunda olsun ve PIND yazmacı ile AND işlemi uygulayarak tablo üzerinden sonuç elde edelim. Birinci tablo PIND, ikinci tablo bizim belirttiğimiz değer sabiti, üçüncü tablo ise sonuç olsun.

1	1	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0

Bu durumda 1 değerini elde ederiz. Eğer belirttiğimiz değerdeki bit konumuna denk gelen ayak “0” olsaydı bu sefer 0 değerini elde edecektik. Diğer ayakların bir (1) veya sıfır (0) olmasının sonuca bir etkisi olmayacaktı. Bu komutu if veya döngü yapıları içerisinde kullanmamız da mümkündür. Aşağıda bir örneğini görebilirsiniz.

```
if(PIND & (1<<1))

{

komutlar;

}
```

Buraya kadar operatörlerle ilgili dijital giriş ve çıkış konuları hakkında anlatılacak konuları anlatmış olduk. Dijital girişte entegre pull-up dirençlerini faal hale getirmeyi ve tersleme komutunu makrolardan sonra anlatacağız. Makroları çok uzun anlatmıza gerek olmayacak. Şimdi makrolara giriş yapalım.

Makrolar

Birazdan bahsedeceğimiz makrolar C dilinin kendisinde olmayıp avr-libc’nin içerisinde olan var avr/sfr_defs.h başlık dosyasında belirtilen makrolardır. Bunlar giriş ve çıkış işlemlerini kolaylaştırmak adına kütüphaneye eklenmiştir. Bu makroları kullanmak yerine operatörleri kullanmak programı daha taşınabilir (portable) yapacaktır. Makroların yaptığı iş aslında operatörlerle yapılan komutun farklı bir biçimde yazılmasıdır. Örneğin _BV() makrosunun tanımı şu şekildedir.

```
#define _BV(bit) (1 << (bit))
```

Yani bizim (1<<1) şeklinde yazdığımız kodu burada _BV(1) olarak yazıyoruz. _BV() makrosunun açılımı “Bit value” yani “bit değeri” anlamına geldiği için programcı için daha anlaşılır oluyor.

_BV() makrosu ile giriş ve çıkış

_BV() makrosunu yukarıda anlattığım için örnek kod vermekle yetineceğiz.

D portunun 1. bitini HIGH yapmak için, **PORTD |= _BV(1);**
D portunun 1. bitini LOW yapmak için, **PORTD &= !_BV(1); ya da PORTD &= ~_BV(1);**
Dijital Okuma Yapmak İçin, **PIND & _BV(1);**

Şimdi bu komutları referans için sözdizimsel olarak tekrar yazalım.

x : Port Adı n: Ayak Numarası

Belli Ayaktan Dijital HIGH çıkış, **PORTx |= _BV(n);**
Belli Ayaktan Dijital LOW çıkış,**PORTx &= ~_BV(n);**
Belli Ayaktan Dijital Giriş, **PINx & _BV(n);**

bit_is_set() ve bit_is_clear() makroları

Bu makrolar adından da anlaşılacağı üzere bit bitin bir (1) yani HIGH mı yoksa sıfır (0) yani LOW mu olduğunu denetlemeye yarar. bit_is_set() ve bit_is_clear() makroları şu şekildedir, **#define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit)) #define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))**

Bu makroların birbirinden tek farkı birinde fazladan tersleme “!” (değil) operatörü olmasıdır. Yani aynı işlemde sonuç bir (1) ise sıfır (0), sıfır (0) ise bir (1) yapmaktadır. Burada makro içinde makroların kullanıldığını görüyoruz. _BV() makrosunu önceden anlattığımız için anlatmaya gerek olmasa da _SFR_BYTE(sfr) makrosu burada yenidir. Bu makro belirtilen adresteki veri hücresinin değerini geri döndürür. Yani PINx dediğimizde PINx yazmacının adresine gider ve oradaki değeri verir. Buna benzeyen _SFR_ADDR() makrosu ise adresi geri döndürür. Bu makrolar konumuzu aştığı için bu kadarıyla yetinelim. Kısacası bit_is_set() ve bit_is_clear() makroları ekstradan bir şey katmayıp bizim operatörler ile yaptığımız işlemi daha anlaşılır hale getirmektedir.

Şimdi bit_is_set() ve bit_is_clear() makrolarına örnek verelim,

bit_is_set(PINC,0) Eğer C portunun 0 numaralı ayağı bir (1) ise bir (1) değerini geri döndürür. Eğer ayak sıfır (0) ise sıfır (0) değerini geri döndürür.

bit_is_clear(PINC,0)

Eğer C portunun 0 numaralı ayağı sıfır (0) ise bir (1) değerini geri döndürür. Eğer ayak bir (1) ise sıfır (0) değerini geri döndürür.

Bu makroları referans için sözdizimsel olarak verelim, **x: Port adı n: ayak numarası**

bit_is_set(PINx,n)

bit_is_clear(PINx,n)

Bu makrolar if gibi karar yapıları içerisinde kullanılırsa operatörlere göre çok daha okunabilir kod yazılabilir. Aşağıda buna benzer bir örnek verilmiştir,

if (bit_is_clear(PINC,0)) { }

Anlatacağımız makrolar buraya kadardı. İsteseydik sadece makrolar üzerinden giriş ve çıkışı anlatıp ne operatörleri ne de makroların iç yapısını anlatma zahmetine girerdik. Fakat bu tarz yaklaşımlar ezberciliğe zemin hazırladığı için en azından başlangıçta seviyeyi çok basit tutmama gereği duyduk. Bundan sonra anlatacağımız iki ufak bilgi kaldı onları da kısaca verelim,

Dahili Pull-UP dirençlerini faal hale getirmek Bunun için yapılması gereken işlem oldukça basittir. Öncelik DDRx yazmacına değer atayarak giriş olarak tanımladığımız portun PORTx yazmacına değer atıyoruz. Örnek aşağıdaki gibidir,

DDRB = 0x00; // PortB nin bütün ayakları giriş yapıldı
PORTB = 0xFF; // PORTB nin bütün ayakları PULL-UP konumunda.

Tersleme / Toggle Yanıp sönen led gibi aynı komutla bir (1) iken sıfır (0) , sıfır (0) iken bir (1) yapmak istiyorsak bu komutu kullanırız. İki farklı kullanımı vardır ikisi de aşağıda verilmiştir. **PORTC = ~PORTC;**

PORTB ^= 0xFF;

Burada PORT veya sabit değer yerlerine operatörler veya makrolarla belli bir ayağı belirtebiliriz. Yukarıda bahsettiğimiz mantık burada da geçerlidir.

Kaynaklar:

<avr/sfr_defs.h>: Special function registers, https://www.programiz.com/c-programming/bitwise-operators, Erişim Tarihi: 18.08.2018

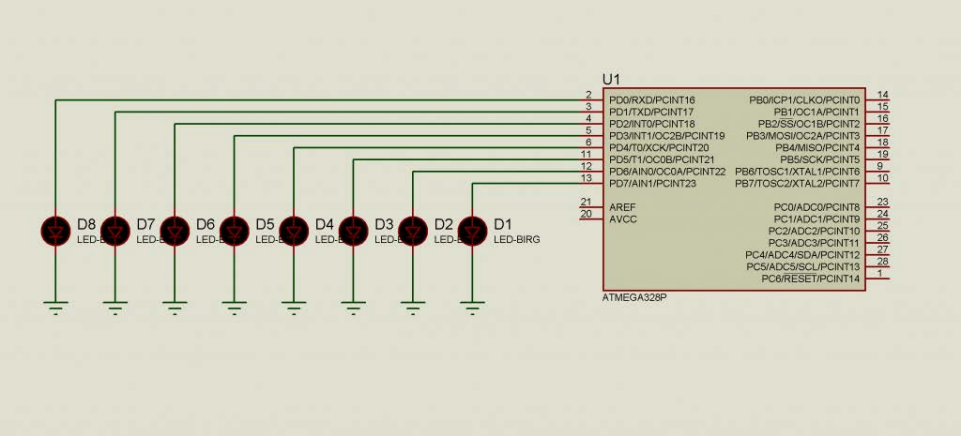
ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 18.08.2018

Karaşimşek ve Düğmeler ile Yürüyen Işık Uygulaması

AVR ile giriş ve çıkış işlemlerini yaparken pek örnek uygulama vermemiştik. Sadece kod sözdizimleri ve tek kod örnekleri üzerinden çalışma mantığını anlatmıştık. Şimdi biraz örnek uygulama yaparak konuyu anlamanıza biraz daha yardım edelim.

Karaşimşek Devresi

Sadece mikrodenetleyicileri değil elektroniği anlatan uygulamalı kitaplarda muhakkak yer edinen bir devre vardır. Bu devreye karaşimşek devresi adı verilir. Sağa ve sola sırayla yürüyen ışık, kimi zaman zamanlayıcı ve entegrelerle kimi zaman ise mikrodenetleyicilerle yapılır. Biz bunu AVR mikrodenetleyiciler için yeniden yazdık. Öncelikle devremizi kuralım ve kodu bütün ayrıntısıyla açıklayalım.



Resme tıklayarak tam çözünürlüklü halini görebilirsiniz.

Şimdi programı yükleyelim ve çalışma prensibini anlatalım.

```
// KARAŞİMŞEK

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{

        DDRD = 0xFF;
        int bekleme = 150;
    while (1)
    {

   for(int i = 0; i < 7; i++)
    {
         PORTD |=_BV(i);
           _delay_ms(bekleme);
         PORTD &=~_BV(i);
        }

        for(int i = 7; i > 0; i--)
        {
                PORTD |=_BV(i);
                _delay_ms(bekleme);
                PORTD &=~_BV(i);
        }
    }
```

```
}
```

#define F_CPU 16000000UL ile işlemcinin saat hızının 16MHz olduğunu belirliyoruz. Böylece delay.h başlık dosyasını çağırdığımızda işlemci hızına göre doğru bekleme sağlanacaktır.

DDRD = 0xFF; ile D portunun bütün ayaklarını dijital çıkış olarak tanımlıyoruz.

int bekleme = 150; bekleme adında bir tamsayı değişkeni tanımlayıp buna "150" değerini atıyoruz. Birazdan bunu delay komutlarında kullanacağız. Böylelikle bekleme süresini bu değişkenin değerini değiştirmekle tüm programda değiştirebiliriz.

for(int i = 0; i < 7; i++) Toplam 8 kere çalışacak bir (0 dahil) bir döngü ayarlıyoruz. 8 kere çalışmasının sebebi portta 8 ayak olması ve buna 8 adet led bağlamamızdan dolayıdır.

PORTD |=_BV(i); For döngüsündeki "i" değişkenini burada ayak numarası olarak kullanıyoruz. For döngüsü i değişkenini 0'dan itibaren 1, 2, 3.. diye artırmaya başladığında biz burada sırayla birinci, ikinci, üçüncü… ayakları seçip bunları yakacağız.

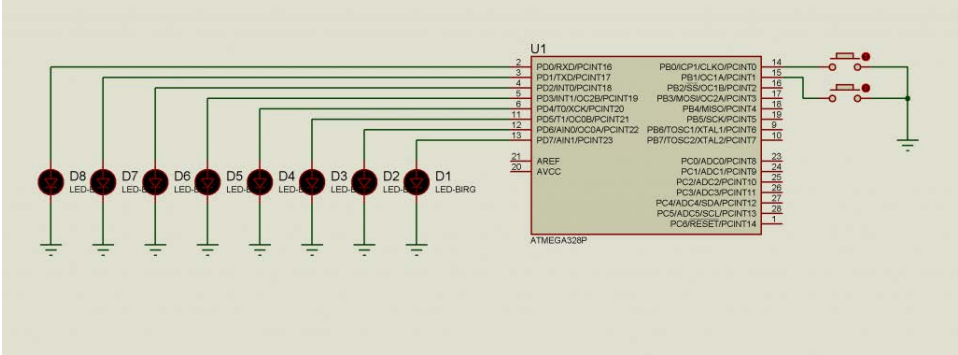
_delay_ms(bekleme); Yukarıda bekleme adında bir değişken tanımlayıp bu değişkene "150" değerini atamıştık. İşte bu değişkenin değerini burada kullanacağız. Komuttan önce bir ayağı HIGH yaparak ledi yakmıştık ve şimdi yanılı halde 150 milisaniye kadar bekleteceğiz.

PORTD &=~_BV(i); Aynı ayağı LOW yapıp ledi söndürmek için bu komutu kullanıyoruz. For döngü yapısının içersindeki bu kod bloğu toplam 8 defa çalıştırılıp ayak numarası sırasında yakıp söndürür.

for(int i = 7; i > 0; i–) Burada yukarıda yaptığımız işlemin tersini yapıyoruz. Yukarıda 0 numaralı ayaktan başlayıp 7 numaralı ayağa kadar yakıp söndürmeye devam etmiştik. Şimdi 7 numaralı ayaktan başlayıp 0 numaralı ayağa kadar yakıp söndürmeye devam edeceğiz ve program tekrar başa alarak çalışmaya devam edecek.

Düğmeler İle Yürüyen Işık

Yaptığımız karaşimşek devresine iki düğme ekleyelim ve farklı bir uygulama yapalım. Bu sefer bir düğmeye basınca sağa diğer düğmeye basınca sola yürüyen ışık uygulaması yapacağız. İki düğme de şaseye bağlanacak ve dahili pull-up dirençleri ile beraber kullanılacak. Öncelikle devre şemasını verelim ardından kodun ayrıntılı açıklamasını yapalım.




```
// Düğmeler ile yürüyen ışık

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{

        DDRD = 0xFF;
        DDRB &=~ ((1<<0) | (1<<1));
        PORTB |= ((1<<0) | (1<<1));
        PORTD |= (1<<PD0);

        while (1)
        {
        if(bit_is_clear(PINB,0))
        {
        PORTD<<=1;
        _delay_ms(250);
        }


        if(bit_is_clear(PINB,1))
        {
                PORTD>>=1;
                _delay_ms(250);
        }
    }
}
```

Programda önceden anlatmadığımız bir ifade yer almakta. Gözümüzden kaçan ifadeyi buradaki kodları anlatırken anlatalım.

DDRB |= ((0<<0) | (0<<1)); Bu kodda iki adet bit ifadesinin yan yana yazıldığı ve aralarında | operatörünün olduğunu görüyoruz. Birden fazla bit ifadesi yazmak istediğimiz zaman parantez içerisinde olmak kaydıyla araya “|” operatörünü koyarak yazabiliriz. Böylelikle daha pratik olur. Burada B portunun 0. ve 1. bitlerini 0 yani giriş olarak ayarlıyoruz.

PORTB |= ((1<<0) | (1<<1)); İfadesinde ise bu iki bitin pull-up dirençlerini bu giriş bitlerini HIGH yaparak aktif hale getiriyoruz. Bu komutla high çıkış alınacağı akla gelmesin çünkü ayakları çıkış olarak değil giriş olarak önceden tanımladık.

PORTD |= (1<<PD0); D portunun 0 numaralı bitini HIGH konumuna getirerek ona bağlı ledi yakıyoruz. Şimdi düğmeler ile bu biti sola ve sağa kaydırmamız gerekli. Sakın en sağdan

ve en soldan taşırmayın

if(bit_is_clear(PINB,0)) B portunun 0 numaralı ayağına basılırsa bu karar yapısının içindeki kod bloğu çalışır.

_delay_ms(250); Düğmeye basılıp işlem yapıldıktan sonra peş peşe aynı işlemi tekrar yapmaması için bir gecikme ekliyoruz. Döngüler ve diğer yapılarla da bu sağlanabilse de şu an için en basit yolunu kullandık.

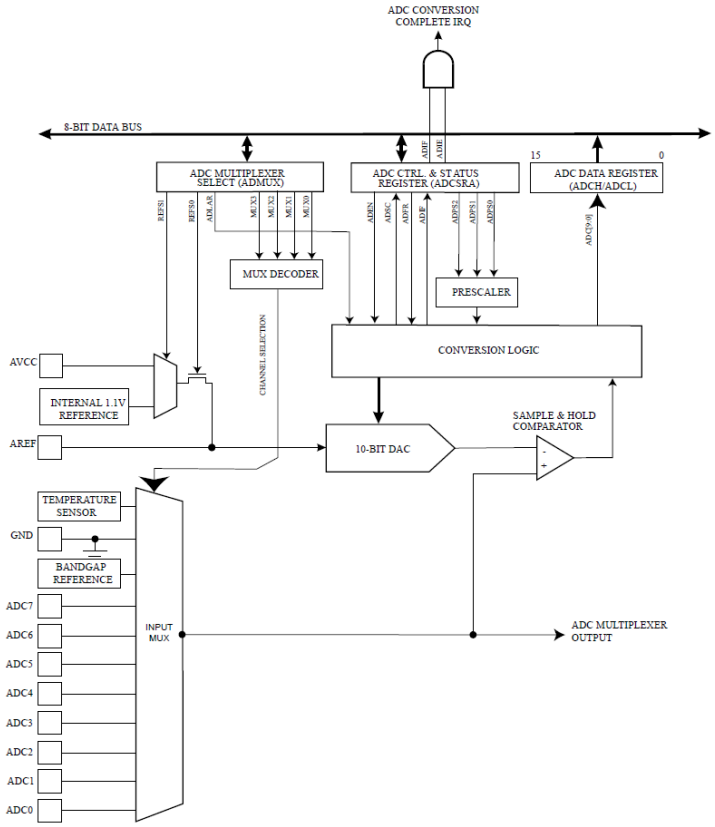
Şu an için bu iki uygulama yeterli olur diye düşünüyorum. Ledler ile animasyon, trafik ışığı gibi pek çok uygulama bu komutlar ile yapılabilse de hemen hepsi aynı prensipte olacağı için her uygulamayı anlatmak gereksiz tekrara sebep olacaktır.

Analog-Dijital Çevirici (ADC)

Dijital olmayan herhangi bir sistem analog sistemdir. Bu bir pil ile lamba devresi de olabilir bir amfide olabilir bir güç kaynağı da olabilir Bunlar mikrodenetleyicilerin anlayacağı dilden konuşmazlar. 1 ve 0 üzerinden işlem yapan bir mikrodenetleyici değişken gerilim değerlerini anlayamaz. Sadece elektriğin belli bir seviyede var olup olmadığını anlar. Dijital veri okumakla değişken gerilim değerlerini ölçmek mümkün değildir. Atmel AVR mikrodenetleyicilerde 10-bit analog-dijital çevirici bulunur ve bu çevirici çoğu projede yeterli hassasiyeti sağlar. Çeşitli seri iletişim yolları üzerinden hassas analog dijital çevrimi yapan modüller ve entegreler olsa da konumuz AVR mikrodenetleyicilerdeki ADC birimini öğrenmek ve bunu kullanmak olduğu için sadece AVR üzerinden anlatmakla yetineceğiz. ADC ile bir termistörün veya ısı sensörünün çıkışını, bir potansiyometreyi, bir gaz sensörünü okuyup bu değeri dijital veriye çevirip işleyebiliriz.

- Atmega328’deki ADC biriminin genel özellikleri
- -10-bit Çözünürlük
- – 13-260 mikrosaniye çeviri zamanı
- -Saniyede 76.9K ölçüm (saniyede 15k ölçüm (10bit))
- – Sıcaklık Sensörü giriş kanalı
- – 0-VCC Giriş gerilimi – Seçilebilir 1.1v referans gerilimi
- – Serbest çalışma veya tek ölçüm modu
- – Ölçüm bitince kesmeye giriş özelliği
- – Uyku modunda gürültü engelleme

Şimdi teknik veri sayfasındaki ADC blok diyagramına bir göz atalım.



Resim: ATmega328P – Microchip Technology , sf. 306, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 18.08.2018

Burada dikkatimizi çeken 3 adet yazmaç bulunuyor. Bu üç yazmaç ile bütün ADC işlemlerini yapıyoruz. ADC için gerilim referans bölümü ise dahili veya harici gerilim referansını belirlemekle görevlidir. ADC'den alınan çıkış ADC Veri Yazmacında bulundurulur. ADC Control ve Status Register (ADC Denetim ve Durum Yazmacı) ise ADC'yi denetleme görevini üstlenir. Geriye kalan ADC multiplexer select (ADC çoğaltıcı seçimi) yazmacı ise hangi ayağın ADC olarak kullanılacağını belirler. Bu yazmaçların her bitinin görevlerini aşağıda açıklayacağız. Bu yazmaçlar anlaşılırsa ADC üzerinde işlem yapmayı anlamış olacağız.

ADC Okuma yapmak için şu adımları yerine getirmek gereklidir.

1. ADC Ayarlanır
2. ADC Faal Edilir
3. ADC Dönüşüm Başlatılır
4. ADC Data Yazmacından ölçülen değer okunur.
   Not: Yazmaçlarda bir bite değer atamak için aşağıdaki maskeleme şablonu kullanılır

YAZMAÇADI |= (BITDEGERI<<BITADI)

Yazmaçadı : Datasheet'den bulacağımız 8 bitlik yazmacın adıdır.

Bitdeğeri: 1 veya 0 değeri.

BitAdı: Yazmacın bitinin özel adı veya bitinin numarası. Bunu da datasheet'den bakacağız.

OKUMA NASIL BAŞLAR?

Okuma PRR (Power Reduction) yazmacının PRADC bitini 1 yaptıktan sonra ADCSRA yazmacının ADSC bitini 1 yaparak başlar. Okuma süreci boyunca bu bit 1 konumunda kalır. Eğer okuma sırasında farklı bir veri kanalı seçilirse adc mevcut kanalı okumaya devam eder ve bitirir. ADC çevrimi otomatik olarak diğer kaynaklar tarafından tetiklenebilir. Otomatik tetikleme ADCSRA yazmacının ADATE bitini HIGH yapmakla faal hale gelir. Tetikleme kaynağı ADCSRB yazmacının ADTS bitine değer atamakla seçilir. Tetik sinyalinde pozitif kenar meydana geldiğinde ADC ön derecelendirici (prescaler) resetlenir ve ölçüm başlar. Bu metod adc dönüşümünü fix interval ile başlatır. Eğer trigger sinyali ölçüm bittiğinde de faal haldeyse yeni ölçüm yapılmaz. Eğer ölçüm sırasında başka bir pozitif kenar tetikleyici sinyalde oluşursa bu sinyaller göz ardı edilir. Bu kesme bayrağı SGREG.I biti kapalı iken veya spesifik kesmeler devre dışı iken bile hazır hale gelir. Bir çevirim kesmeye sebep olmadan da yapılabilir. Bunun için kesme bayrağı her yeni çevrimde sıfırlanmalıdır. ADC kesme bayrağını tetikleyici kaynağı yapmak çevrim biter bitmez yeni çevrim yapmaya sebep olur. Serbest çalışma modu denilen bu yöntem ile sürekli adc okuması yapılabilir. İlk ölçüm ADCSRA. ADSC bitini 1 yaparak başlamak zorundadır.

PRESCALER

ADC frekansının sağlıklı ölçüm yapabilmesi için 50-200kHz aralığında olması gerekir. Bu aralıkta düşükfrekanslar daha yüksek doğruluk verir. ADC prescaler'i CPU frekansına göre prescaler bitleriyle ayarlanmalıdır. ADCSTA ADPS bitleriyle bu ayarlanır . Prescaler ADCSRA.ADEN (Adc Enable) biti ile çalışır ve ADCEN biti 0 olana kadar çalışmaya devam eder ve ardından resetlenir.

ADC Yazmaçları

Yukarıda eski notlarımdan kopyaladığım biraz farklı dilde olan bir parafgraf vardı. Eğer yeterince karışık bulunduysa şimdi giriş ve çıkış işlemlerinde olduğu gibi önce yazmaçları anlatmakla işe başlayalım. ADC için üç yazmaç kullanıldığını önceden söylemiştik. Bu yazmaçlar üzerinde yapılan doğru işlem ve okuma ile biz bir bacağa giden analog sinyalin dijital veriye dönüştürülmüş halini elde edebiliyoruz. Onun için öncelikle yazmaçları tanımamız kodları anlamamız için muhakkak gereklidir. Şimdi yazmaçları sırasıyla anlatalım.

ADMUX Yazmacı

Yazmacın görüntüsü aşağıdaki gibidir. Tıklayarak büyütebilirsiniz.

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Bit 7:6 – REFSn Referans Seçimi

Yazmacın REFS0 ve REFS1 adındaki bitleri analog ölçümdeki gerilim referansını seçmeye yarar. Bu bitlerin konumları ve bu konumların getirdiği özellik aşağıdaki tablodaki gibidir.

REFS [1:0]	Gerilim Referansı Seçimi
00	AREF, İç Gerilim Referansı Kapalı
01	AVcc AREF ayağındaki harici kondansatör ile
10	Rezerve (Kullanım dışı)
11	İç 1.1V gerilim AREF ayağındaki harici kondansatör ile

Görüldüğü gibi biz gerilim referansını besleme (5V), Dahili 1.1V ve AREF ayağında uygulanacak gerilim olarak üç ayrı şekilde seçebiliyoruz. Arduino'da bu analogReference() fonksiyonu ile oluyordu. Biz burada doğrudan mikrodenetleyicinin donanımına müdahale ediyoruz. Örneğin gerilim referansını AVcc yani besleme olarak seçmek istersek şöyle bir komut yazmalıyız.

ADMUX |= (1<<6); ADMUX &= ~(1<<7);

Bu komut ADMUX yazmacının 6. bitini bir (1) 7. bitini ise sıfır (0) yapar. Tabloda 01 ile belirtilen AVcc referansı sağlanmış olur.

Bit 5 ADLAR (ADC Sola Hizalama)

ADLAR biti ADC Veri Yazmacındaki dönüşüm sonucunu sola hizalamaya yarar. Bu bit HIGH konumda sonuç sola hizalı iken normalde sonuç sağa hizalıdır. ADLAR bitinin değeri değişince doğrudan hizalama işlemi yapılır.

Bit 3:0 – MUXn (n = 3:0) – Analog Kanal Seçimi

Bu bitler hangi ayaktan analog okumayı yapacağımızı seçmeye yarar. Sekiz kanal analog olduğundan bahsetmiştik. İşte bu bitlerin konumlarını değiştirerek bu kanalları seçiyoruz. Mesela ADC0 ayağından okuma yapmak istiyorsak MUX bitlerini 0000 yapmamız gerekir. Aşağıdaki tabloda MUX bitlerinin konumu ve bunların sonucu verilmiştir.

MUX [3:0]	Giriş
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Sıcaklık Algılayıcısı
1001	Rezerve
1010	Rezerve
1011	Rezerve
1100	Rezerve
1101	Rezerve
1110	1.1V (Vbg)
1111	0V (GND

ADCSRA Yazmacı (ADC Kontrol ve Durum Yazmacı)

Yazmacın görüntüsü aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – ADEN : ADC Faal Bu biti HIGH yapmak ADC'yi faal hale getirir. Sıfır yapınca ADC kapatılır. ADC'yi dönüşüm aşamasında kapatmak dönüştürmeyi sonlandırır.

Bit 6 – ADSC: ADC Dönüşüm Başlatımı

Tek dönüştürme modunda her bir çevirimi başlatırken bu biti HIGH yaparız. Serbest çalışma modunda ise ilk çevirimi yapmak için bu biti HIGH yapmamız gerekir. Çevirim yapmadan önce ADEN bitini HIGH konumuna getirmek sonra ADSC bitini HIGH konuma getirmek gerekir. Bunun ardından Analog-Dijital Çevirimi yaparız. ADSC biti bu çevirim boyunca HIGH konumunda kalır ve çevirim bittikten sonra tekrar LOW konumuna geçer. Bu bite sıfır değerini yazmanın bir işlevi yoktur.

Bit 5 – ADATE : ADC Otomatik Tetikleyici Faal

Bu bit HIGH yapılırsa, ADC'nin otomatik tetikleyici faal hale gelir. ADC çevirime seçili tetikleyici sinyalinin yükselen kenarında başlar. Tetikleme kaynağı ADC Tetikleme Seçim Bitleriyle seçilir. Bu bitler ADTS olup ADCSRB yazmacındadır.

Bit 4 – ADIF : ADC Kesme Bayrağı

Bu bit ADC çevirimi bittikten ve veri yazmaçları güncellendikten sonra HIGh konumunda olur. ADC çevirim bitme kesmesi, ADIE biti ve SREG yazmacındaki I biti HIGH konumda ise yürütülür. ADIF donanım tarafından eğer uyan bir kesme kullanma vektörü kullanılırsa LOW konumuna geçer. Bunun yerine ADIF bitine LOW yazmakla da bu bayrak sıfırlanır.

Bit 3 – ADIE: ADC Kesme Faal

Bu bit HIGH yapılırsa ve SREG kesmesindeki I biti de HIGh konumdaysa ADC Çevirim Bitme Kesmesi faal hale gelir.

Bit 2:0 – ADPSn : ADC Ön derecelendirici seçimi

Bu bitler sistem saat frekansı ile ADC'nin giriş saati arasındaki bölme faktörünü belirlemeye yarar.

ADPS[2:0]	Bölme Faktörü
000	2
001	2
010	4
011	8
100	16
101	32

110	64
111	128

Buraya kadar ADC hakkında bilgi verip yazmaçların bir kısmını tanıttık. Sıradaki yazıda yazmaçların geri kalanını anlatıp örnekler üzerinden devam edeceğiz.

Kaynaklar

ATmega328P – Microchip Technology , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 18.08.2018 Dökmetaş, Gökhan, “Arduino Eğitim Kitabı”, İstanbul, 2016

Analog-Dijital Çevirici (ADC) Yazmaçları

Önceki yazımızda analog ve dijital çeviriciyi anlatmıştık ve en son yazmaçlarda kalmıştık. Şimdi geri kalan yazmaçları anlatalım ve analog-dijital çeviriciyi nasıl kullanacağımızı anlatmakla devam edelim.

ADCL – ADC Veri Yazmacı (Düşük Bit)

Bit	7	6	5	4	3	2	1	0
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

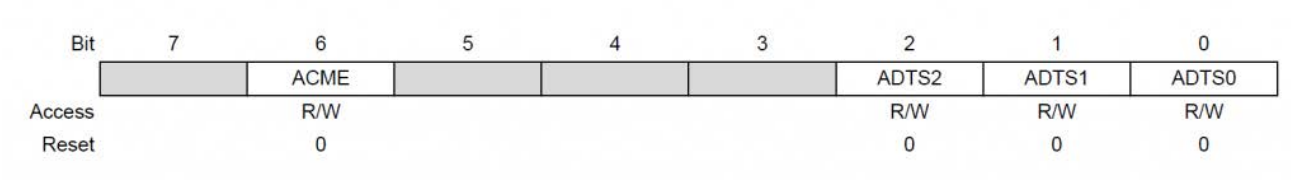
ADC çevirimi bittiği zaman sonuç bu iki yazmaç içerisinde bulunur. ADC çevirimi 10-bit olmasından dolayı 8-bitlik yazmaçlara sığmamakta ve yazmacın birine 8-bit veri ötekisine ise kalan 2 bitlik veri kaydedilir. O yüzden iki yazmacı da okuyup bu verileri birleştirerek ADC çevirim sonucunu elde ederiz. Bu yazmaçlar sadece okunabilir yazmaçlar olup ADLAR bitinin (önceki yazıda bahsettik) durumuna göre sola veya sağa hizalı şekilde verileri depolar. Bitleri tek tek açıklamaya gerek yok çünkü 0-7 arası tüm bitler ADC çevirim sonucunu barındıran bitlerdir. Bu bitleri okuyup birleştirerek son işlemi yapmış olacağız.

ADCH – ADC Veri Yazmacı (Yüksek Bit)

Bit	7	6	5	4	3	2	1	0
							ADC9	ADC8
Access							R	R
Reset							0	0

Bu yazmaç okunan ADC değerinin son 2 bitini (soldan sağa ilk iki biti) içinde saklar. Bu iki yazmaç veri okuma yazmacıdır ve sadece veri okumaya yarar. ADLAR bitine göre sola hizalandığında bu yazmaç 8-bitlik veri bulundurup diğer yazmaç son iki bitinde (soldan iki bit) kalan veriyi barındırır.

ADCSRB – ADC Denetim ve Durum Yazmacı B



Bit	7	6	5	4	3	2	1	0
		ACME				ADTS2	ADTS1	ADTS0
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

Bu yazmaç ADC denetim ve ayar bitlerini bulunduran yazmacın devamı niteliğinde üzerinde bazı ayar bitlerini barındırır.

Bit 6 – ACME : Analog Karşılaştırıcı Faal

Bu bit HIGH yapıldığı zaman ve ADC kapatıldığı zaman (ADCSRA yazmacındaki ADEN biti sıfır olduğunda) ADC çoklayıcısı analog karşılaştırıcıya negatif giriş seçer. Eğer bu bit sıfır olarak yazılırsa AIN1 ayağı analog karşılaştırıcı için negatif giriş olarak seçilir.

Bit 2:0 – ADTSn : ADC Otomatik Tetikleyici Kaynağı [n = 2:0]

ADCSRA yazmacı içindeki ADATE biti HIGH yapıldıysa bu değerler ADC çevirimi için tetikleyici kaynağını belirler. Eğer ADATE sıfır olarak ayarlanırsa ADTS bitleri bir işleve sahip olmaz. Çeviriim seçili kesme bayrağının yükselen kenarında tetiklenir. Eğer ADCSRA içindeki ADEN biti HIGH yapılırsa çevirim başlar. Serbest Çevirim modunda ADC kesme bayrağı HIGH konumda olsa dahi tetiklenme olmaz. Bu bitlerin görevleri aşağıdaki tablodaki gibidir.

ADTS [2:0]	Tetikleme Kaynağı
000	Serbest Çalışma Modu
001	Analog Karşılaştırıcı
010	Dış Kesme İsteği 0
011	Zamanlayıcı/Sayıcı0 karşılaştırma örtüşmesi A
100	Zamanlayıcı/Sayıcı0 Taşma
101	Zamanlayıcı/Sayıcı1 Karşılaştırma Örtüşmesi B
110	Zamanlayıcı/Sayıcı1 Taşma

111	Zamanlayıcı/Sayıcı1 Yakalama Olayı

DIDR0 – Dijital Giriş Devredışı Bırakma Yazmacı

Bit	7	6	5	4	3	2	1	0
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bu yazmaca yazılan HIGH bitleri karşılık gelen ADC kanalındaki dijital girişi devre dışı bırakır.

Buraya kadar yazmaç kısmını anlatmış olduk. Şimdi bu yazmaçlar üzerinde nasıl ADC ölçümünün yapıldığını anlatmaya sıra geldi. Bunu da sonraki yazıda anlatacağız.

ADC Kullanımı ve Örnek Kodlar

Önceki yazılarda ADC birimini anlatmış ve ADC'nin görevlerinden bahsetmiştik. Teknik veri kitapçığında yer alan yazmaçları anlatmış ve bu yazmaçların ve yazmaçlardaki bitlerin görevlerini ayrıntılı açıklamıştık. Şimdi ise C dilinde bu yazmaçları nasıl kontrol ederek ADC birimini çalıştıracağımızı anlatalım.

ADC işlemini adım adım anlatmak gerekirse şu şekilde olmalıdır,

- Seçili ADC kanal ayağını giriş olarak tanımlamalıdır.
- AVR'nin ADC modülü faal hale getirilmelidir. Çünkü açılış resetinde güç tüketimini önlemek için devre dışıdır.
- Çevirim hızı seçilir. ADPS2:0 bitleri bu işlemi yapar.
- ADC giriş kanallarındaki gerilim referansı seçilir. ADMUX yazmacındaki REFS0 ve REFS1 bitleri referans seçmek için görevlidir. ADMUX yazmacındaki MUX4:0 bitleri ise ADC giriş kanalını seçmeye yarar.
- ADCSRA yazmacındaki ADSC biti HIGH yapılarak çevirim başlatılır.
- ADCSRA yazmacındaki ADIF yani ADC kesme bayrağı biti kontrol edilerek çevirimin bitmesi beklenir.
- ADIF biti HIGH konuma geçtikten sonra ADCL ve ADCH yazmaçlarındaki dijital veri çıkışı okunur. Öncelikle ADCL yazmacını sonrası ADCH yazmacını okumak gereklidir.
  İşte AVR mikrodenetleyicilerde ADC okuma işlemleri bu şekilde olmaktadır. Elimizin altında bir makina var ve bu makinayı talimatına uygun kullanıp analog değerden dijital değeri elde etmemiz gereklidir. Bu talimatları üretici bize bildirmektedir ve bunun mantığını ve iç yapısını bu noktadan sonra anlatmamız pek mümkün değildir. Çünkü üretici mikrodenetleyici ve çevre birimlerinin bizim için gerekli kısmını teknik veri sayfasında paylaşmaktadır. Daha alt seviyeye inerek konu dışına çıkmış oluruz. İnşallah ileride mikrodenetleyici mimarisini ve dijital elektroniği anlattığımız derslerde buna değineceğiz. Şimdilik yazmaç boyutuna insek dahi biraz ezbere işlem yapmak zorundayız.

Şimdi elimdeki örnek kodlardan dersi anlatmaya devam edelim. C dilinde oluşturduğumuz iki fonksiyonumuz var. Bunlardan biri ADC'yi başlatan ve diğeri ise ADC çevirimi yapıp sonucu geri döndüren asıl fonksiyon olarak görev yapıyor. Öncelikle adc_init() fonksiyonunu yazmaçlar üzerinden anlatarak nasıl çalıştığını anlatalım.

```
void adc_init(void){
 ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));   // 125Khz ADC referans saati
 ADMUX |= (1<<REFS0); // Referans AVCC yani 5V
 ADCSRA |= (1<<ADEN); // ADC'yi Aç
 ADCSRA |= (1<<ADSC); // İlk deneme ölçümünü yap ve diğer ölçüme hazır hale
getir.
}
```

ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)); Bu komutta ADCSRA yazmacının ilk üç biti olan ADPS0, ADPS1 ve ADPS2 bir (1) yapılır. ADC saati için ön derecelendirici ayarı ile 16Mhz/128 yani 125KHz ADC referans saati ayarlanmış olur. İşlemci 16MHz'de çalışırken ADC 50kHz ile 200kHz arası çalışmaktadır. O yüzden işlemcinin saat hızı ön derecelendirici (prescaler) ile bölünmelidir. Yüksek frekans hızlı ölçüm fakat düşük doğruluk verir. Düşük frekans ise yavaş ölçüm ve yüksek doğruluk. İhtiyaca göre bölme oranı belirlenmelidir.

ADMUX |= (1<<REFS0); ADC'nin hangi gerilim değerini esas alarak ölçüm yapacağını belirler. ADC yazmaçlarını anlatırken 1.1V, Harici ya da Besleme olarak üç ayrı referans gerilimi alabileceğinden bahsetmiştik. Burada REFS0 bitini bir (1) yaparak referans gerilimini besleme gerilimi yapıyoruz. Yani 0-5V arası 0-1023 farklı değer elde edeceğiz.

ADCSRA |= (1<<ADEN); ADC'yi açıyoruz. ADC açılıştan itibaren kapalı olup boş yere çalışmaz. O yüzden ADC'yi kullanmak üzere şimdi açıyoruz. Bu bit bir makinanın açma kapama düğmesi gibi düşünülebilir. ADEN bitini bir (1) yaparak ADC'yi çalışır hale getirdik.

ADCSRA |= (1<<ADSC); ADSC biti bir (1) yapılarak bir deneme çevirimi başlatılır. Buradaki amaç birimi bir sonraki çevirime hazırlamaktır. İlk çevirim yavaş olduğu için ilk çevirimi burada yaptık.

```
uint16_t read_adc(uint8_t channel){
 ADMUX &= 0xF0; // Eski kanal bilgisini temizle
 ADMUX |= channel; // Yeni kanal bilgisini yükle
 ADCSRA |= (1<<ADSC); // Yeni Çevirim Başlat
 while(ADCSRA & (1<<ADSC)); // Çevirim bitene kadar bekle (bu kısım çok önemli)
 return ADCW; // ADC çevirim değerini geri döndür.
}
```

ADMUX &= 0xF0; ADMUX yazmacının gerekli kısımlarını bir sonraki kanal bilgisi yazılmak üzere temizliyoruz. Her çevirimden önce hangi ayaktan ADC bilgisinin okunacağı belirlenmelidir. Bunu da fonksiyon byte değerinden (uint8_t) argüman alarak bir sonraki kodda işletecektir.

ADMUX |= channel; ADMUX yazmacına kanal numarası(yani ayak numarası 0, 1, 2.. vd.) yazılır. Böylelikle hangi ayaktan analog okuma yapacağımızı belirlemiş oluruz.

ADCSRA |= (1<<ADSC); ADCSRA yazmacındaki ADSC biti bir (1) yapılarak yeni çevirim başlatılır. Artık çevirimin bitmesini ve değer okumayı bekleyeceğiz.

while(ADCSRA & (1<<ADSC)); ADCSRA yazmacındaki ADSC bayrak biti çevirim bitince bir (1) konumuna geçer. O yüzden bunun bir olmasını bekleyene kadar mikrodenetleyiciyi döngüye sokuyoruz.

return ADCW; Çevirim bittiğini bayrak bitinden öğrendikten sonra ADCW yani ADCH ve ADCL yazmaçlarının toplamı olan veriyi değer olarak döndürüyoruz. ADCW'in açılımı ADC Word demektir yani 16-bit veriden oluşur. Bu mikrodenetleyicinin teknik veri sayfasında verilmese de kodda yer almaktadır. Bu olmasaydı önce ADCL yazmacındaki veriyi okuyup sonra da ADCH yazmacındaki veriyi 8-bit sola kaydırıp ADCL yazmacındaki veriyi attığımız word tipindeki değişkene atabilirdik. ADC verisinin atılacağı değişken muhakkak 8 bitten büyük olmalıdır. Bu int (word) olabilir.

ADC hakkında anlatacaklarımızın hepsi bu kadarla sınırlı olmasa da bu kadarı şimdilik yeterlidir. Geri kalanı ilerleyen derslerde başka konular içinde veya ayrı bir konu olarak anlatabiliriz. Şimdilik ADC hakkında yeterli bilgiyi verdiğimize inanıyoruz.

Elimizde ADC ile ilgili birkaç AVR kütüphanesi vardı. Bu kütüphanelerin arasından birini bütün ayrıntılarıyla inceleyerek hem ADC'nin çalışma prensibini tekrar gözden geçireceğiz hem de size yararlı bir kütüphane vermiş olacağız. Bu kütüphanenin yazarını ve bağlantısını şimdi bulma imkanım olmadığı için bir referans veremeyeceğim. adc.h ve adc.c olarak iki dosyanın tüm kodlarını elimizden geldiği kadarıyla size açıklayacağız ve çalışma mantığını bir daha anlatacağız. Burada bir daha anlatmak istemem önceki kodların çok da yeterli gelmeyişinden dolayıdır. Şimdi çalışmamıza başlayalım.

adc.h dosyası

```
#define MCU_FREQ 16000000UL
uint16_t convert;
uint16_t loop_count;

void     initADC( void );
uint16_t  ADC10bit( uint8_t channel );
uint16_t  ADC08bit( uint8_t channel );
void     DELAY_US( uint16_t microseconds);
```

Bu dosyada kullanılacak fonksiyonların ve değişkenlerin tanımları yapılmıştır. .h uzantılı dosya bizim kütüphaneyi kullanmak için müracaat edeceğimiz ilk dosya olmalıdır. .c dosyası kullanıcıyı pek alakadar etmeyen ama kütüphanenin asıl bölümünü oluşturan dosyadır. Öncelikle .h dosyası üzerinden tanımlanan fonksiyon ve değerleri anlatarak .c dosyasındaki fonksiyonların anlaşılmasına yardımcı olalım.

#define MCU_FREQ 16000000UL , Bu kütüphanede MCU_FREQ adındaki değerin mikrodenetleyicinin saat hızı olduğunu anlıyoruz. Demek ki bu kütüphane 16MHz'de çalışan bir mikrodenetleyici ile kararlı çalışabiliyor. Biz kullanıcı olarak bu değeri değiştirerek kullanmamız gerekir. AVR kütüphanelerinde sık sık göreceğiniz gibi kütüphaneyi olduğu gibi kullanmak pek mümkün değildir. Bunun bir sebebi ise kütüphanelerin C dilinde yazılmış olmalarından kaynaklanır. Arduino kütüphaneleri C++ dilinde yazılıp sınıf-nesne ilişkisinden dolayı gerekli parametreleri sınıf içerisinde nesne oluştururken yazabiliyorduk. C dilinde yazılan bir kütüphanede ise bu parametreler başlık dosyasında yer alıp doğrudan başlık dosyasından değiştirilmeye ihtiyaç duyar.

uint16_t convert; Burada convert (dönüştürme) adında 16 bitlik işaretsiz tam sayı değişken tanımlanmış. 16 bit olmasından dolayı bunun çevirim sonucunu barındıracak değişken olduğunu çok rahat tahmin edebiliriz. ADC'nin nasıl çalıştığını anlamamız başkalarının yazdığı kodu ilk okumada anlamamızı sağladı. Bunun doğru olup olmadığını ise .c uzantılı dosyada göreceğiz. Şimdi devam edelim.

uint16_t loop_count; Burada yine 16 bitlik yani "word" cinsinde işaretsiz bir tamsayı değişken tanımlanıp adı loop_count (döngü sayısı) olarak belirlenmiş. Tahminimize göre bu kütüphane kendi içerisinde delay fonksiyonu kullanıyor ve bu delayda kaç kere döngünün çevrildiğine dair bilgiyi bu değişken içerisinde barındırıyor. .c dosyasına henüz bakmadığımız için yine kabaca bir tahminde bulunuyoruz.

void initADC(void); Bu fonksiyon önceki yazıda kullandığımız fonksiyona benzer bir işleve sahip olmalıdır. ADC'yi açıp hazırlamak için kullanılacak fonksiyon olduğunu "init" yani "initialize" (başlatmak) ifadesinden anlıyoruz.

uint16_t ADC10bit(uint8_t channel); uint16_t ADC08bit(uint8_t channel); Bu iki fonksiyonun aldığı argüman ve döndürdüğü değerden ADC okuma fonksiyonu olup adlarından birinin 10-bit okuma diğerinin ise 8-bit okuma yapacağını anlıyoruz.

void DELAY_US(uint16_t microseconds); Bu fonksiyon ise adından ve aldığı argümandan anlaşıldığı üzere bekleme işlemini yapıyor.

İşte kullanıcının okuyup anlaması gereken ve buna göre kullanması gereken .h başlık dosyasından .c dosyasını okumadan anladığımız bu kadardı. Bazı kütüphanelerde .h başlık dosyasında fonksiyonların işlevi ve kütüphane kullanımı hakkında bilgi yer almaktadır. Bu bilgiler kütüphaneyi anlamamıza yardımcı olsa da her zaman bu tarz kütüphanelerle karşılaşmak pek mümkün değildir. Kütüphane referansı ya da kullanma talimatı yoksa iş başa düşmektedir. Şimdi .c dosyasını inceleyelim.

adc.c dosyası

```
#include <stdint.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "adc.h"
void initADC( void )
{
  ADCSRA = ( 1 << ADEN  ) | ( 1 << ADSC  )
         | ( 1 << ADPS2 ) | ( 1 << ADPS1 );
  while ( ADCSRA & ( 1 << ADSC ) );
}

uint16_t ADC08bit( uint8_t channel )
{
    /* set for 8-bit results for the desired channel number then start the
       conversion; pause for the hardware to catch up */

    ADMUX  = ( 1 << ADLAR ) | ( 1 << REFS0 ) | channel;
    ADCSRA = ( 1 << ADEN  ) | ( 1 << ADSC  );
    DELAY_US( 64 );
```

```
    /* wait for complete conversion and return the result */

    while ( ADCSRA & ( 1 << ADSC ) );

    return ADCH;
}

uint16_t ADC10bit( uint8_t channel )
{
    ADMUX  = ( 1 << ADLAR ) | ( 1 << REFS0 ) | channel;
    ADCSRA = ( 1 << ADEN  ) | ( 1 << ADSC  );
    DELAY_US( 64 );
    while ( ADCSRA & ( 1 << ADSC ) );
        convert = ADCH;
        convert <<= 2;
        convert += (ADCL >> 6);
        return convert;
}

void DELAY_US( uint16_t microseconds )
{

#if MCU_FREQ == 8000000UL

    /* 8mhz clock, 4 instructions per loop_count  */
    loop_count = microseconds * 2;

#elif MCU_FREQ == 1000000UL

    /* 1mhz clock, 4 instructions per loop_count */
    loop_count = microseconds / 4;

#elif MCU_FREQ == 16000000UL

    /* 1mhz clock, 4 instructions per loop_count */
    loop_count = microseconds / 4;

#else
#error MCU_FREQ undefined or set to an unknown value!
    loop_count = 0; /* don't really know what to do */
#endif

    __asm__ volatile (
        "1: sbiw %0,1" "\n\t"
        "brne 1b"
        : "=w" ( loop_count )
        : "0"  ( loop_count )
    );
}
```

Bu dosya biraz uzun olduğu için parça parça inceleyeceğiz. Yukarıda dosyanın tamamına bir göz gezdirmeniz yeterli. Şimdi ilk fonksiyonla başlayalım.

```
void initADC( void )
{
  ADCSRA = ( 1 << ADEN  ) | ( 1 << ADSC  )
         | ( 1 << ADPS2 ) | ( 1 << ADPS1 );
  while ( ADCSRA & ( 1 << ADSC ) );
}
```

Bu fonksiyon ADCSRA yazmacının bazı bitlerine yazma işlemi yaparak işe başlar. Eğer teknik veri kitapçığından yazmaçları öğrenmeseydik burada nasıl bir işlem yapıldığı hakkında bir fikrimiz olmayacaktı. Fonksiyonun ilk komutunda ADCSRA'nın ADEN, ADSC, ADPS2 ve ADPS1 bitleri bir (1) yapılır. Bu komutun sonucu özetle şöyle olacaktır. ADC Faal hale getirilecektir sonrasında ADC'nin hazır olması için ilk dönüşüm başlatılacaktır ve ADC'nin ön derecelendiricisi 64 olarak seçilecektir. 16MHz'de çalışan bir işlemci için bu 250kHz demektir. İkinci komutta ise ADCSRA yazmacındaki ADSC yani ADC çevriminin bittiğini belirten bayrak biti 1 olana kadar program döngüde kalacaktır. Burada yapılan en önemli işlem ADEN bitinin bir (1) yapılması ve ADC'nin çalışır hale gelmesi ve ADC ön derecelendiricisinin ayarları yapılıp ADC'nin düzgün çalışmasının sağlanmasıdır.

```
uint16_t ADC08bit( uint8_t channel )
{
    /* set for 8-bit results for the desired channel number then start the
       conversion; pause for the hardware to catch up */

    ADMUX  = ( 1 << ADLAR ) | ( 1 << REFS0 ) | channel;
    ADCSRA = ( 1 << ADEN  ) | ( 1 << ADSC  );
    DELAY_US( 64 );

    /* wait for complete conversion and return the result */

    while ( ADCSRA & ( 1 << ADSC ) );

    return ADCH;
}
```

Bu fonksiyon argüman olarak işaretsiz 8 bit tamsayı türünde (uint8_t) channel adında bir değer alır. Bu değer tabi ki analog okumanın yapılacağı kanalın numarası olacaktır. 8-bit değer döndürse de fonksiyon uint16_t yani işaretsiz 16-bit tamsayı değer döndürür. Bunun sebebi değeri atayacağımız değişkenin genellikle 16-bit olmasından dolayıdır. Şimdi ilk komutu inceleyelim.

ADMUX = (1 << ADLAR) | (1 << REFS0) | channel; Soldan itibaren açıklamaya başlarsak öncelikle ADMUX yazmacı üzerine bir değer atama söz konusu olduğunu söyleyebiliriz. ADLAR biti bir (1) yapılarak değerin sola hizalı olması sağlanır. Bunun sebebi değerin ilk 8 bitlik parçası okunup geriye kalan 2 bitlik teferruat kısmı okunmayacaktır. Böylelikle değeri kabaca okumuş oluyoruz. ADMUX yazmacının REFS0 biti 1 yapılır böylelikle analog gerilim referansı besleme gerilimi (5V) olarak seçilmiş olur. Burada hemen şu yorumu yapmamız mümkündür, bu kütüphanenin eksik yanlarından biri .h dosyasında referans gerilimi seçmek için bir seçenek verilmemiştir ve kullanıcı referans gerilimini değiştirmek için kodlarla oynamak zorundadır. En son olarak da channel değişkenine atadığımız değer ADMUX yazmacının son 3 bitine etki edecektir. Eğer bu fonksiyonun argüman kısmına aşırı bir değer yazarsak tüm bu ayarlar bozulacaktır. Çünkü argümandan gelen değer bir engelle karşılaşmadan doğrudan yazmaca gitmektedir. Bunu engelleyici bir kod yazılmamıştır.

ADCSRA = (1 << ADEN) | (1 << ADSC); Bu fonksiyon ADCSRA yazmacının ADEN bitini bir (1) yaparak analog çeviriciyi eğer kapalıysa tekrar açar ve ADSC yazmacını bir (1) yaparak analog çevirimi başlatır.

DELAY_US(64); Analog çevrim için bir bekleme süresi verilmiştir.

while (ADCSRA & (1 << ADSC)); Bu komut analog çevirimin bittiğini kontrol etmek amacıyla ADCSRA yazmacındaki ADSC bitinin bir (1) olup olmadığını kontrol ederken mikrodenetleyiciyi döngüya sokar. Analog-Dijital Çevirici mikrodenetleyicinin işlemcisinden bağımsız çalıştığı için işlemci döngüdeyken analog-dijital çevirici işlemciden bağımsız çalışmaya devam eder. İşlemci bu bayrak bitini kontrol ederek işlemin bittiğinden haberdar olur.

return ADCH; 8 bitlik ADCH yazmacındaki değer fonksiyonda döndürülür. Böylelikle 8 bitlik analog ölçüm işlemi bitmiş olur. Şimdi 10-bit ADC çevriminin nasıl yapıldığına bakalım.

```
uint16_t ADC10bit( uint8_t channel )
{
    ADMUX  = ( 1 << ADLAR ) | ( 1 << REFS0 ) | channel;
    ADCSRA = ( 1 << ADEN  ) | ( 1 << ADSC  );
    DELAY_US( 64 );
    while ( ADCSRA & ( 1 << ADSC ) );
        convert = ADCH;
        convert <<= 2;
        convert += (ADCL >> 6);
        return convert;
}
```

ADMUX = (1 << ADLAR) | (1 << REFS0) | channel; Bu komut yukarıdaki fonksiyonun aynısı. Burada da yine değeri sola hizalıyor.

ADCSRA = (1 << ADEN) | (1 << ADSC); Öteki fonksiyonun aynısı olan bu komut ile ADC başlatılıyor. Alttaki iki fonksiyon da işlemin aynısı olduğu için bir daha anlatma gereği duymuyoruz.

convert = ADCH; Burada “convert” değişkeninin nerede tanımlandığını önceden okumuş olduğumuz .h dosyasından hatırlayabiliriz. .h dosyasının .c dosyasından önce okunması gerektiğini buradan çıkarabiliriz. uint16_t tipinde değişken olan “convert” burada analogdan dijitale çevirilen değeri saklayacaktır. Öncelikle ADCH yazmacındaki değerler convert değişkenine aktarılıyor.

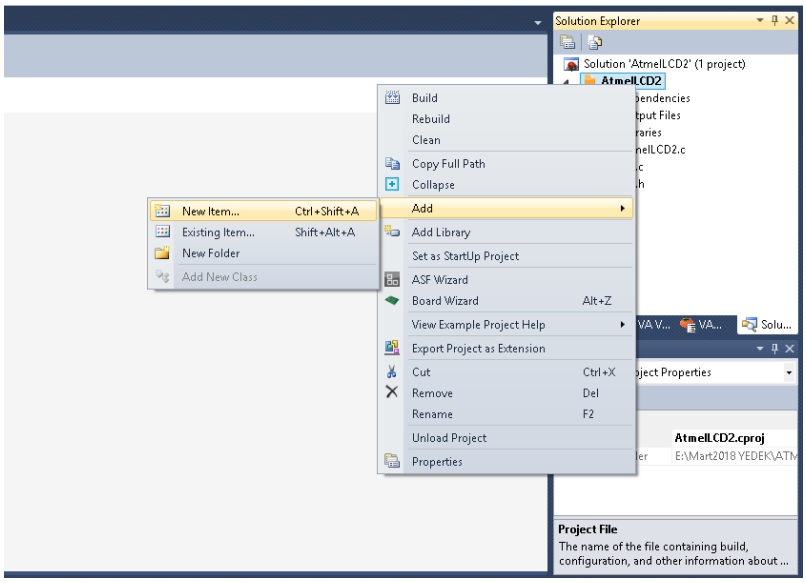
convert <<= 2; Burada convertteki tüm bitler iki bit sola kaydırılıyor. ADCH yazmacındaki bit örneğin “11111111” şeklinde olup “convert” değişkenine aktarıldığında ise “0000000011111111” şeklinde olacaktır. Burada sağda iki yer açmak için bunu iki adım sola kaydırırsak sonuç şu şekilde olacaktır, “0000001111111100”.

convert += (ADCL >> 6); Burada ADCL yazmacındaki tüm bitler 6 adım sağa kaydırılıyor. Bunun sebebi yazmacın son iki bitinde kalan veri bulundurulup bunun da normalde son bit verileri olmasından dolayıdır. Örneğin ADCL biti “11000000” ise bu işlemden sonra “00000011” olacaktır. Bu değer de convert değişkeninin içindeki boş bitlere aktarılacaktır. Böylelikle convert değişkeni 10 bitlik sonuçla doldurulmuş olacaktır. Böyle bit kaydırma işleminin yapılması değerlerin sola hizalanmasından dolayıdır. Bu da bu kütüphaneyi yazan kişinin tercihidir.

return convert; Burada 16 bitlik convert değeri fonksiyondan döndürülür.

```
void DELAY_US( uint16_t microseconds )
{

#if MCU_FREQ == 8000000UL

    /* 8mhz clock, 4 instructions per loop_count  */
    loop_count = microseconds * 2;

#elif MCU_FREQ == 1000000UL

    /* 1mhz clock, 4 instructions per loop_count */
    loop_count = microseconds / 4;

#elif MCU_FREQ == 16000000UL

    /* 1mhz clock, 4 instructions per loop_count */
    loop_count = microseconds / 4;

#else
#error MCU_FREQ undefined or set to an unknown value!
    loop_count = 0; /* don't really know what to do */
#endif

    __asm__ volatile (
        "1: sbiw %0,1" "\n\t"
        "brne 1b"
        : "=w" ( loop_count )
        : "0"  ( loop_count )
    );
}
```

Burada yazarın kullandığı bekleme fonksiyonunun kendisi görülmektedir. loop_count bekleme ölçüsünü belirlemede MCU_FREQ değerini esas almaktadır. Kütüphane yazarı 3 adet MCU_FREQ değeri belirlemiştir. Yani bu kütüphane bu hali ile .h dosyasındaki MCU_FREQ değiştirilse dahi 8MHz, 10MHz ve 16MHz hızlarında kullanılabilir. Burada yazar bunun farkında olup #error komutu ile bu üç frekanstan başka bir frekans değeri giren kullanıcıya hata mesajını iletmiştir. Fonksiyonun aşağısında ise Assembly dilinde bekleme komutu yer almaktadır. Kütüphaneyi incelediğimizde bazı zayıflıklar söz konusu olsa da bunu inceleyerek kütüphanelerin nasıl bir yapıda olduğunu satır satır size gösterdik. Şimdi ise Atmel Studio'da projemize nasıl kütüphane dosyalarını ekleyeceğimizi gösterelim.



Yukarıdaki resimdeki gibi **Solution Explorer** penceresinde projeye sağ tıklayıp **Add / Existing Item** diyoruz ve dosya seçimi ekranından .h ve .c uzantılı dosyaların ikisini de seçiyoruz.

Projemizde ise baş kısma **#include "kutuphaneadi.h"** şeklinde yazıyoruz. Bundan sonra bu kütüphaneye ait fonksiyonları kullanabiliriz.

Kütüphaneyi kullanmak isteyenler aşağıdaki bağlantıdan indirebilir.
https://github.com/GDokmetas/AVR-ADC-Library

USART İletişim Protokolü ve Birimine Giriş

Artık ADC ile ilgili öğreneceklerimizin çoğunu öğrendik ve yeni bir konuya giriş yapmamızın vakti geldi. ADC hakkındaki bilgilerin tamamı olmasa da büyük çoğunluğunu anlatmış olduk. Geriye kalan ADC kesmesi, dahili sıcaklık algılayıcısı ve diğer konuları ileride ihtiyaç duyduğumuz zaman anlatacağız. Şimdi Evrensel Senkron Asenkron Alıcı Verici yani USART birimini ve bunun nasıl kullanılacağını anlatarak derslerimize devam edelim.

Bu derse başlamadan önce şunu söylememizde fayda var. Bu dersler elektroniğe ve gömülü sistemlere yeni başlayanlar için anlaşılmaz gelebilir. En azından C dili, temel elektronik ve mikrodenetleyicilerde bir temelinizin olması gerekiyor. Bu dersler tam olarak Arduino'da uzmanlaşmış, çeşitli proje yapmış ve bir ileri adıma geçmek isteyenlere hitap etmektedir. O yüzden bu konularda yeni olanlar Arduino'ya başlayıp Arduino konuları ile beraber temel elektroniği ve C programlamayı da öğrenmelidir. Arduino üzerinde iyi bir seviyeye gelen biri için bu dersler zor gelmeyecektir. Yazdığım "Arduino Eğitim Kitabı" tam da bu amaç için yazılmıştır. Bu kitap elektronik ve yazılıma giriş yapanlarda sağlam bir temel oluşturma görevini üstlenecektir.

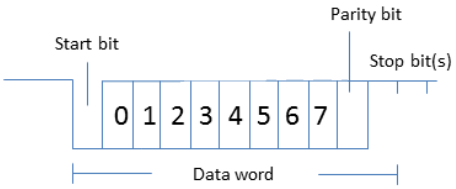
UART ve USART

Bu protokollerin ikisi birbirine karıştırılabilmektedir. Aslında prensip olarak birbirine çok benzemekte fakat birinde veri asenkron diğerinde ise senkron olarak iletilmektedir. Bu senkronizasyon SPI protokolünde olduğu gibi CLOCK (Saat) frekansı ile sağlanmaktadır. UART protokolü bu yüzden USART'a göre daha basit kalmaktadır. Eğer bir UART protokolünde veri göndermek istersek sadece tek bir ayağı kullanmamız ve verinin bitlerini sırasıyla belli bir zamana göre göndermemiz yeterli olacaktır. Bu belli bir zamanı karşı tarafın anlaması için bir saat frekansı olmadığı için sıkıntı çıkabilmektedir. Bunun için baud rate adı verilen veri iletişim hızını iki aygıtta da belirtmemiz gerekir.

Baud rate saniyede gönderilen bit sayısı olarak belirtilir. Örneğin bir aygıtın veri iletişim hızı 9600 ise bu aygıt saniyede 9600 bit gönderebilir. Buna karşılık alıcı aygıtın da 9600 bit veriyi aynı hızda okuması gerekir. Eğer aynı hız sağlanamazsa veriler düzgün okunamaz. Bu yüzden senkronizasyon ayarını bizim iki aygıtta da önceden yapmamız gerekir.

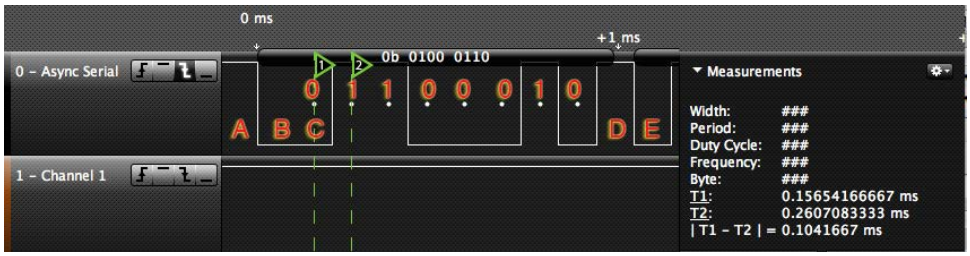
UART protokolü gönderim için bir ayak ve alım için bir ayak kullanır. Gönderilen veri karşı taraftaki aygıtın alım bacağına (RX > TX) alınan veri ise karşı taraftaki aygıtın gönderim bacağına (TX > RX) bağlanmak zorundadır. İki taraflı iletişim için iki hatta ihtiyacımız olsa da tek taraflı iletişim için iki hattın bağlanma zorunluluğu yoktur.

UART protoklünde veriler seri bir halde tek bir bacaktan gönderilir. Bunun öncesinde Başlama biti ve en sonunda ise eşitlik (sağlama) biti vardır. UART veri okumaya başlama bitinin okunmasıyla başlanır. Aşağıdaki resimde örnek bir UART iletişiminin mantıksal şeması vardır.



Resim:
https://cms.edn.com/ContentEETimes/Images/15%20Quinnell/Bloggers/UART1.png

Şimdi örnek bir UART iletişiminin lojik analizör görüntüsünü ayrıntılı olarak inceleyelim. Bu resmi anladıktan sonra UART protokolü için anlaşılmadık bir nokta kalmayacaktır.



Resim: http://www.gammon.com.au/images/Async_logical_analyzer_2.png

A : Burada seri iletişim başlamamıştır ve seri iletişim ayağı HIGH konumdadır. Bu ayak HIGH konumda olduğu sürece de seri iletişim başlamaz.

B: Burada seri iletişim ayağı LOW konumuna çekilerek başlama biti alıcıya gönderilir ve seri iletişime hazır olması gerektiği söylenir. Başlama biti 1.5 saat dalgası kadar uzun kalmalıdır. Normal veri bitleri için 1 saat dalgası yeterlidir.

C: Burada sıralı olarak ilk bitten başlayarak (sağdan itibaren) byte tipindeki 8 bitlik verinin bitleri sırayla gönderilir. Bit bir (1) ise HIGH konumda sıfır (0) ise LOW konumda bir saat dalgası kadar bekler.

D: Stop Bitidir. Burada okumanın bittiği haber verilir.

E: Bir sonraki bayt verisinin başlangıç bitidir.

Bu seri iletişim bazen RS-232 olarak da adlandırılsa da RS-232 protokolünün gerilim değerleri ile mikrodenetleyicinin UART protokolünün gerilim değerleri birbirinden farklıdır. Bu farklılıktan dolayı bir mikrodenetleyici bilgisayarın RS-232 portuna doğrudan bağlanamaz. RS-232 portu olan bilgisayar pek kalmasa da bağlamak isteyenler MAX232 entegresini kullanmalıdır.

Eşlik (Parity) Bitleri

Bu bitleri kullanmak tercihe bağlı olup verinin doğru iletilip iletilmediğini sağlamak için kullanılır. Bu bitler iletilen verinin bitlerinin toplamına göre tek veya çift olur. (Tek = 1 Çift = 0) Bu bite göre verinin doğru olarak iletilip iletilmediğine karar verilir.

USART ise bunun üzerine asenkron iletişimden kaynaklanan senkronizasyon sorununu gidermek için ek bir Saat frekansı hattına sahiptir. AVR ve Arduino projelerinde genellikle UART kullanılmakta olup bu dezavantajına rağmen oldukça stabil sonuçlar alınmaktadır. Bu dersi burada bitirip AVR'nin USART birimine sonraki derste geçelim.

USART Özellikleri ve Yazmaçlara Giriş

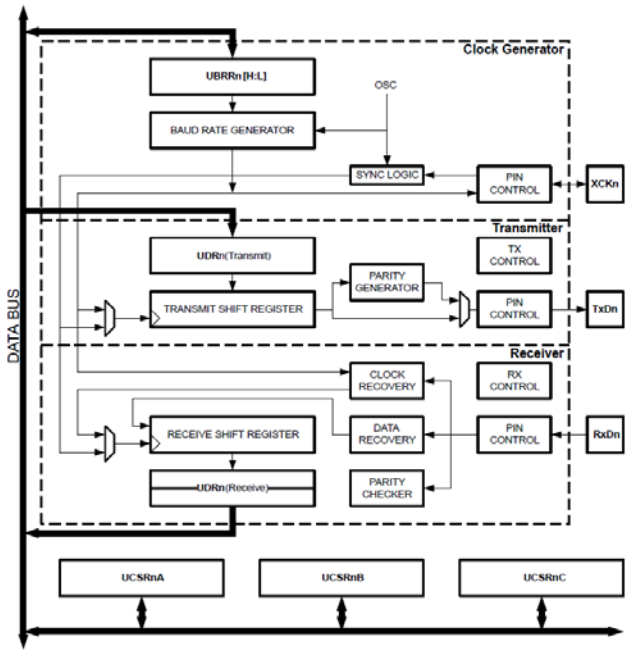
ADC birimini yazmaçlar üzerinden anlattığımız gibi USART birimini de yazmaçlar üzerinden anlatacağız. Yazmaçlar yapısı itibariyle "şu şu ile yarar, bu bu işe yarar" şeklinde anlatılıp

ezberci bir anlatımla bitirilse de yazmaçlar öğrenildikten sonra bütün kodları ve kütüphanelerin çalışma prensibi çok rahat anlaşılabilir. Yazmaçlar öğrenilmediği durumda kodları ve kütüphane fonksiyonlarını anlamadan ezberlemek zorunda kalırız bu durum da ezbere ve anlamadan kod yazmaya sebep olur. Günümüzde ezbere kod yazan programcıların aşırı derecede fazla olmasının sebebi kaynakların ezbere ve yüzeysel bir anlatımla konuyu geçiştirmesinden dolayıdır. Bu anlatımın yüzeysel olmasının en büyük sebebi yabancı olan kaynakların Türkçe'ye aktarımında kırpma ve özensiz Türkçeleştirme yapılmaktadır. Örneğin yabancı dilde 2-3 sayfada anlatılan bir Arduino projesi Türkçe'ye aktarılırken iki paragrafta sığ bir dille anlatılıp kod ve şemadan ibaret ve bilgi verme yönünden oldukça yetersiz bir hale geliyor. Bu projeyi ezbere yapmak öğrenmeye bir katkı sağlamaz. Orjinal içerik yazılması olmazsa olmaz şartlardan değil. Türkçeleştirilmiş bir yabancı içeriğin düzgün ve eksiksiz Türkçeleştirilmesi bile Türkçe kaynak açısından büyük bir ilerleme olacaktır. Buradan içerik yazarlarına tavsiyemizi verdikten sonra Atmega328P entegresinin USART biriminin özelliklerini anlatmaya başlayalım.

USART Özellikleri

- Tam Dubleks Çalışma (Bağımsız Seri Alıcı ve Verici Yazmaçları)
- Asenkron veya Senkron (Eş zamanlı) çalışma
- Ana ya da Uydu Saat sinyali ile senkron çalışma
- Yüksek çözünürlüklü veri iletim hızı jeneratörü
- 5, 6, 7, 8 ya da 9 Data biti desteği ve 1 ya da 2 stop biti desteği
- Tek ya da Çift Eşitlik oluşturma ve donanım destekli eşitlik biti denetleme.
- Data OverRun Algılaması
- Framing hatası Algılaması
- Yanlış başlangış biti algılaması ve dijital düşük geçiş filtresi ile gürültü filtreleme.
- TX bitişi, TX Veri yazmacı boş ve RX bitişi olarak üç ayrı kesme özelliği
- Çoklu işlemci iletişim modu
- Çift hızlı asenkron iletişim modu

Blok Diyagramı



Resim: ATmega328P – Microchip Technology , sf. 226 , http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 18.08.2018

Diyagramı incelediğimizde üç ana bölüme ayrıldığını görüyoruz. Clock Generator (Saat sinyali üretici), Trasmitter (Verici), Receiver (Alıcı) olarak üçe ayrılan bu bölümlerden hariç bazı yazmaçların da veri yoluna bağlandığını görüyoruz. Ayrıca sağ kenarda XCKn, TxDn, RxDn olarak adlandırılan üç adet bölüm mikrodenetleyicide yer alan seri iletişim ayaklarını temsil ediyor. Buradan USART'ın dış dünya ile bağlantısının bu üç ayaktan sağlanacağını çıkarabiliriz. Bu bacakların mikrodenetleyici kılıfındaki kaç numaralı ayağa denk geldiğini önceki yazılarda verdiğim mikrodenetleyicinin ayak haritasından görebilirsiniz.

Saat sinyali üretici hem XCKn ayağından saat frekansı çıkışında hem de alıcı ve verici ünitelerinin ölçüm yapması için veri okuma hızının belirlenmesinde görevlidir. Saat sinyali üretecinin UBRRn yazmacı ile ayarlandığını görüyoruz.

Verici ünitesinde ise UDRn yazmacı dikkatimizi çekiyor. Verici ünitesindeki bu tek yazmaç veri yoluna bağlıdır. Alıcı ünitesinde ise yine tek yazmaç bulunup veri yoluna bağlıdır. Buradan bunların veri alışverişinde kullanılacak yazmaçlar olduğunu anlıyoruz.

Bundan başka UCSRnA, UCSRnB ve UCSRnC adında üç ayrı yazmaç da veri yoluyla USART bloğuna bağlı görünüyor. Bu yazmaçların görevlerini de aşağıda açıklayacağız.

UDR0 – USART Giriş ve Çıkış Veri Yazmacı

Bu yazmaç USART veri iletiminde tampon bellek (buffer) görevini görür. Alıcı ve verici veri yazmaçları aynı adresteki giriş ve çıkış yazmacını paylaşır ve bu yazmaç da UDR0 olarak adlandırılır. Verici Veri Tampon Yazmacı (TXB) UDR0 yazmacına yazılacak veriyi hedef alacaktır. UDR0 yazmacını okumak da alıcı veri tampon yazmacının içeriğini bize verecektir. Verici tampon verisi ancak UCSR0A yazmacındaki UDRE0 bayrak bitinin bir (1) olması ile yazılabilir. Verici tamponuna veri yazıldığında verici faal hale gelmiş olur. Verici değiştirme yazmacına (Transmit Shift Register) boş olduğu zaman verici veriyi yükler. Sonrasında veri seri olarak TxD0 ayağından gönderilir. Yazmacın görüntüsü şu şekildedir.

Bit	7	6	5	4	3	2	1	0
	TXB / RXB[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – TXB / RXB[7:0]: USART Transmit / Receive Data Buffer

Burada gönderim ve alım için ortak kullanılan ve okuma/yazma yapılabilen 8 bitlik bir yazmaç görüyoruz. Yapacağımız okuma ve yazmalar bu yazmaç üzerinden olacaktır. Diğer yazmaçlar ise iletişimin nasıl olacağına dair ayarları içeren yazmaçlardır. Yazmaçların

fazlalığından biraz karmaşık görünse de prensipte oldukça basittir. Şimdi diğer yazmaçları anlatmakla devam edelim.

UCSR0A – USART Denetleme ve Durum Yazmacı 0 A

Bit	7	6	5	4	3	2	1	0
	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

Bit 7 – RXC0 : USART Veri alımı tamamlandı

Bu bayrak biti tampon bellekte okunmamış bir gelen veri olduğunda bir (1) konumuna geçer. Tampon bellekte veri kalmadığı zaman ise tekrar sıfır (0) konumuna geçer. Ayrıca Seri veri alım kesmesini gerçekleştirmek için de bu bit kullanılır. Kısaca bu bayrak bite bize verinin gelip okunmayı beklediğini haber verir.

Bit 6 – TXC0 : USART Veri Gönderimi Tamamlandı

UDR0 yazmacındaki tüm bitler gönderildiği zaman bu bit bir (1) konumuna geçer. TXC0 bayrak biti eğer gönderim tamamlanma kesmesi yürürse otomatik olarak sıfır (0) konumuna geçer. Ayrıca elle de bu biti sıfırlama imkanımız vardır. Bu bayrak biti USART gönderim tamamlanma kesmesini yürütmek için kullanılabilir.

Bit 5 – UDRE0 : USART Veri Yazmacı Boş

Bu bit gönderici tamponunun yani UDR0'ın yeni veri almaya müsait olduğunu belirtir. Eğer UDRE0 biti bir (1) konumunda ise tampon bellek boştur. Ayrıca bu bayrak biti Veri yazmacı boş kesmesini yürütmek için kullanılabilir.

Bit 4 – FE0 : Çerçeve Hatası

Bu bit alıcı tamponunun sonraki karakteri çerçeve hatasına sahipse bir (1) konumuna geçer.

Bit 3 – DOR0: Veri Aşma

Bu bit veri aşma tespit edilirse bir (1) konumuna geçer. Bir veri aşma hatası alış tamponu dolu olduğunda gerçekleşir.

Bit 2 – UPE0: USART Eşlik Hatası

Bu bir eşlik hatası gerçekleştiğinde bir (1) konumuna geçer.

Bit 1 – U2X0: USART Çift Gönderim Hızı

Bu bit ancak asenkron çalışmada etkili olur. Senkron iletişimde bu bit bir (1) olmalıdır. Bu biti bir (1) yapmak veri aktarım hızı bölücüsünü 16'dan 8'e düşürür ve böylelikle transfer hızını ikiye katlar.

Bit 0 – MPCM0: Çoklu işlemci iletişim modu

Bu bit çoklu işlemci iletişim modunu faal hale getirir.

Sonraki yazımızda geri kalan yazmaçları açıklayacağız ve ardından örnek kodlar üzerinden iletişimin nasıl gerçekleştiğini anlatacağız. Teknik veri kitapçığından anlatmak oldukça yorucu olsa da bunları anlamadıkça kodları anlamak mümkün olmuyor. Aksi halde kod örneklerini verip bunları kopyalayın kullanın demekle yetinecektik. O yüzden gerçekten öğrenmek istiyorsanız sabırla dersleri okuyup anlamanız gereklidir.

USART Yazmaçları

Bu dersimizde Atmega328P mikrodenetleyicisinin USART yazmaçlarını anlatmaya kaldığımız yerden devam ediyoruz. Yazmaçları anlattıktan sonra örnek kodlar üzerinden anlatmaya devam edeceğiz. Dersleri yazmaya başladığımda belli bir sıra gözetmesem de artık belli bir düzene oturduğunu görüyoruz. Diğer konuları da aynı düzende anlatıp hepsini bitirdikten sonra ise uygulamalarla devam edeceğiz. Şimdi kaldığımız yerden devam edelim.

UCSR0B – USART Denetim ve Durum Yazmacı 0 B

Bit	7	6	5	4	3	2	1	0
	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – RXCIE0 : Alım Tamamlandı Kesmesi Etkinleştirme 0

Bu bite bir (1) yazmak RXC0 bayrak bitindeki kesmeyi faal hale getirir. USART Alım Tamamlandı kesmesi ancak RXCIE0 bitinin bir (1) yapılmasıyla gerçekleşir. Ayrıca kesme için SREG'deki Genel Kesme Bayrağı (Global Interrupt Flag) ve UCSR0A'daki RXC0 biti de bir (1) yapılmalıdır.

Bit 6 – TXCIE0 : Gönderim Tamamlandı Kesmesi Etkinleştirme 0

Bu bite bir (1) yazmak TXC0 bayrak bitindeki kesmeyi etkin hale getirir. USART gönderim tamamlanma kesmesi ancak TXCIE0 bitinin bir (1) yapılmasıyla, SREG yazmacındaki

genel kesme bayrak bitinin bir (1) yapılmasıyla ve UCSR0A yazmacındaki TXC0 bitinin bir (1) yapılmasıyla yürür.

Bit 5 – UDRIE0 : USART Veri Yazmacı Boş Kesmesini Etkinleştirme 0

Bu biti bir (1) yapmak UDRE0 bayrak bitindeki kesmeyi etkin hale getirir. Veri yazmacı boş kesmesi ancak UDRIE0 bitinin bir(1) yapılmasıyla, SREG'deki genel kesme bayrak bitinin bir (1) yapılmasıyla ve UCSR0A yazmacındaki UDRE0 bitinin bir (1) yapılmasıyla yürür.

Bit 4 – RXEN0 : Alıcı Etkinleştirme 0

Bu biti bir (1) yapmak USART alıcısını etkinleştirir.

Bit 3 – TXEN0 : Verici Etkinleştirme 0

Bu biti bir (1) yapmak USART vericisini etkinleştirir.

Bit 2 – UCSZ02 : Karakter boyutu 0

UCSZ02 biti UCSZ0 [1:0] bitleriyle beraber çalışır. Bu bitler alıcı ve verici görevleri yürütülürken belirlenen veri boyutu çerçevesini belirlemede kullanılır.

Bit 1 – RXB80 : Alıcı 8. Veri Biti

Eğer veri çerçevesi boyutu 9 bit ise burada 9. bit saklanır. UDR0'ın düşük bitlerini okumadan önce okunmalıdır.

Bit 0 – TXB80 : Verici 8. Veri Biti

Eğer veri çerçevesi boyutu 9 bit olarak belirlenirse gönderilecek 9. bit buraya yerleştirilir. UDR0'ın düşük bitlerini yazmadan önce yazılmalıdır.

UCSR0C – USART Denetim ve Durum Yazmacı 0 C

Bit	7	6	5	4	3	2	1	0
	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 / UDORD0	UCSZ00 / UCPHA0	UCPOL0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

Bit 7:6 UMSEL0n : USART Modu Seçimi 0 [n = 1:0]

Bu bitler USART'ın çalışma modunu seçer. Modlar aşağıdaki tablodaki gibidir.

UMSEL0 [1:0]	Mod
00	Asenkron USART
01	Senkron USART
10	Rezerve
11	Ana SPI (MSPIM)

MSPIM etkinleştirildiğinde UDORD0, UCPHA0 ve UCPOL0 bitleri aynı yazma işleminde bir (1) konumuna alınabilir.

Bit 5:4 – UPM0n : USART Eşlik Modu 0 [n = 1:0]

Bu bitler eşlik üretimi tipini ayarlar ve bunu denetler. Eğer etkinleştirilirse verici otomatik olarak eşlik bitini oluşturur ve veri çerçevesi içerisinde bunu gönderir. Alıcı eşlik değerini oluşturur ve gelen veri ile UPM0 ayar kısmında karşılaştırır. Eğer eşliksizlik tespit edilirse UCSR0A yazmacındaki UPE0 bayrağı bir (1) konumuna getirilir.

UPM0[1:0]	Eşlik Modu
00	Kapalı
01	Rezerve
10	Etkin, Çift Sayı
11	Etkin, Tek Sayı

Bit 3 – USBS0 : USART Stop biti seçimi 0 Bu bit verici tarafından yerleştirilen stop bitlerinin sayısını belirler. Alıcı bu ayarı görmezden gelir.

USBS0	Stop Biti
0	1-bit
1	2-bit

Bit 2 – UCSZ01 / UDORD0 : USART karakter boyutu / veri sırası

UCSZ0 [1:0] USART modunda: UCSZ0[1:0] bitleri UCSR0B yazmacındaki UCSZ02 biti ile beraber kullanılır. Bu bitler veri çerçevesindkei veri bitlerinin sayısını belirler. Aşağıdaki tabloda işlevi açıklanmıştır.

UCSZ0[2:0]	Karakter Boyutu
000	5-bit

001	6-bit
010	7-bit
011	8-bit
100	Rezerve
101	Rezerve
110	Rezerve
111	9-bit

UDPRD0 : Ana SPI modu : Eğer bir (1) yapılırsa son bitten itibaren veri gönderilir. Eğer sıfır (0) yapılırsa baş bitten itibaren veri gönderilir. SPI modundaki USART'a sonra geleceğimiz için bu bilgi şimdilik gerekli değildir.

Bit 1 – UCSZ00 / UPCHA0 : USART karakter boyutu / Saat evresi

UCSZ00 : Usart modunda UCSZ01'i referans eder.

UCPHA0: Ana SPI modunda eğer veri XCK0'ın ilk veya son kenarında örnekleniyorsa bunun ayarını yapar.

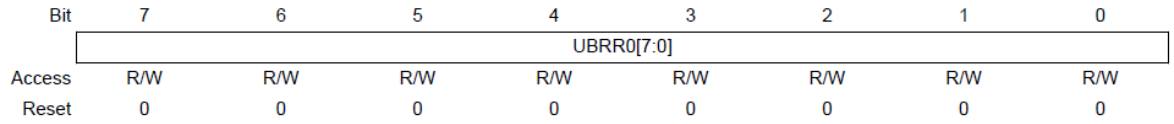
Bit 0 – UCPOL0 – Saat Frekansı Kutbu 0

USART0 Modunda: Bu bit sadece senkron modda kullanılır. Asenkron mod kullanılacaksa bu bir sıfır yapılmalıdır. UCPOL0 biti veri çıkışındaki değişiklik ve veri girişi örneklemesindeki ilişki ve senkron saatini ayarlamaya yarar (XCK0) USART saat frekans kutbu ayarı aşağıdaki gibidir.

UCPOL0	Gönderilen Veri Değişti (TxD0 ayağındaki çıkış)	Alınan veri örneklendi (RxD0 ayağındaki giriş)
0	Yükselen XCK0 Kenarı	Alçalan XCK0 Kenarı
1	Alçalan XCK0 Kenarı	Yükselen XCK0 Kenarı

Ana SPI Modu: UCPOL0 biti XCK0 saatinin kutbunu ayarlar. UCPOL0 ve UCPHA0 bitlerinin beraber kullanımı veri transferindeki zamanlamayı ayarlar.

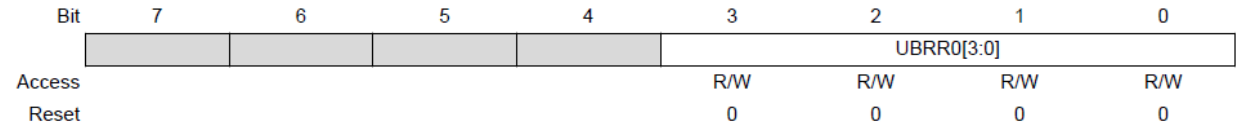
UBRR0L – USART Baud Oranı 0 Yazmacı – Düşük



Bit 7:0 – UBRR0 [7:0] USART Baud Oranı (Veri iletim hızı)

Bu 12 bitlik yazmaç USART veri iletim hızı bilgisini içinde bulundurur. UBRR0H baş dört biti bulundururken UBRR0L son sekiz biti bulundurur. Baud hızı iletişim sırasında değişirse Alıcı ve verici düzgün işleyemez. UBRR0L yazmacına veri yazmak baud oranında ani bir değişme yapar.

UBRR0H – USART Baut Oranı 0 Yazmacı – Yüksek



Bu yazmaçta UBRR0L yazmacının geri kalan baş bitleri (most significant bits) saklanır.

USART birimi için yazmaçlar bu kadardır. Bunları anlatmak oldukça yorucu olsa da eksiksiz anlatmaya çalıştık. Bir sonraki yazımızda örnek kodları inceleyerek çalışma prensibini ayrıntılı olarak anlatacağız. Biz yazmaçların tamamının işlevini ezberlemenizi istemiyoruz. Yazmaçlar ve özellikleri hakkında bilginiz olsun yeterli.

USART Kullanımı ve Örnek Kod İncelemesi

AVR'nin USART yazmaçlarını önceki yazımızda bitirmiştik. Artık bu yazmaçlar üzerinden nasıl program yazılacağı ve USART biriminin nasıl kullanılacağını anlatalım. Yazmaçların ADC birimine göre biraz daha ayrıntılı olması ve farklı iletişim tiplerinin ortak kullandığı bitler olması biraz kafamızı karıştırabilir. USART prensipte çok basit olduğu için sadece bayt alma ve bayt gönderme olarak kısa fonksiyonlar yazarak bunu gerçekleştirmemiz mümkündür. Aynı ADC biriminde olduğu gibi öncelikle USART birimini hazır hale getirmek için bir fonksiyon yazmak zorundayız. Bu fonksiyonu yazarak işe başlayalım.

USART birimi yüklenmesi için şu adımlar gerçekleştirilmelidir,

1. Baud Oranı Ayarlanmalıdır
2. Veri Boyutu Ayarlanmalıdır
3. TXEN ve RXEN bitleri ile Alıcı ve Verici Etkinleştirilmelidir.
4. Eşlik biti ve Stop bitlerinin sayısı ayarlanmalıdır.

Şimdi bunu program kodu üzerinden görelim.

```
#define BAUD 9600                                   // Baud Oranını Tanımlıyoruz
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)            // UBRR için baud verisini
ayarlıyoruz


void uart_init (void)
{
    UBRRH = (BAUDRATE>>8);                      // Baudrate'in üst bitlerini
kaydırıyoruz
    UBRRL = BAUDRATE;                           // kalan bitleri yazdırıyoruz
    UCSRB|= (1<<TXEN)|(1<<RXEN);                // Alıcı ve vericiyi
etkinleştiriyoruz
    UCSRC|= (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);   // Veri formatını ayarlıyoruz
}
```

Bu örnek kodda her komutun ne iş gördüğünü açıklamada yazsak da komutlara bakarak anlamak için bunu yazmaçlar üzerinden anlatmak lazımdır. Öncelikle yukarıda tanımladığımız makroların işlevinden bahsedelim. Yukarıda baud 9600 olarak belirlenmiştir. Bunu anlamak kolay olsa da BAUDRATE olarak tanımlanan makrodaki matematik işlemi bize yeni görünecektir. Mikrodenetleyicinin teknik veri sayfasında UBRR yazmacına yazılacak baud oranının hesabına dair bir formül mevcuttur. Bu kod bu formülün kod haline dönüşmüş şeklidir. Merak edenler için aşağıda formülü verelim.

Operating Mode	Equation for Calculating Baud Rate(1)	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{16(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{16\mathrm{BAUD}} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{8(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{8\mathrm{BAUD}} - 1$
Synchronous Master mode	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{2(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{2\mathrm{BAUD}} - 1$

Kısaca açıklarsak F_CPU değerinde bulunan işlemcinin saat hızı ile BAUD değerinin 16'ya çarpımı bölünür ve bundan 1 çıkarılır. Delay fonksiyonlarında olduğu gibi F_CPU tanımını yapmamışsak yapmamız gerekir. Örneğin #define F_CPU 16000000UL komutu işlemci saat hızının 16MHz olduğunu belirtir.

UBRRH = (BAUDRATE>>8);

Bu kod elde edilen BAUDRATE değerinin bitlerini 8 bit sağa kaydırarak ilk bitleri 8 bitlik ölçüde açığa çıkarır. Örneğin elimizde 12 bitlik bir "0000111111111111" değeri olsun. Bunu sekiz bitlik bir yazmaca yüklediğimizde sağdaki sekiz bit yüklenecek fakat soldaki sekiz bit yüklenemeyecektir. Bunu Yüksek ve Düşük yazmaçlarına sığdırmak için öncelikle sağdaki sekiz birlik düşük yazmaç değerini atmamız gerekir. Bu işlemden sonra değerimiz "0000000000001111" olacaktır. Böylelikle bu dört bit yüksek yazmacın ilk dört bitine

yazılacaktır. Geri kalan değer ise bir sonraki komutta düşük yazmaca yazılacaktır. Elimizdeki mikrodenetleyici 8-bit olduğu için böyle büyük değerleri sığdırmak zorunda kalıyoruz.

UBRRL = BAUDRATE; Burada düşük yazmaca BAUDRATE'in tamamı yazdırılmış gibi görünse de aslında sağdaki sekiz biti yazdırılmıştır. Geri kalan bitler de yazdırıldığına göre baud oranı değerini yazdırma ve bu oranı ayarlama işlemi bitmiş oluyor.

UCSR0B|= (1<<TXEN)|(1<<RXEN); UCSRB yazmacındaki TXEN ve RXEN bitleri bir (1) yapılır. Bu aynı ADC'de ADEN bitinin bir (1) yapılması gibi alıcı ve vericiyi etkinleştirir.

UCSR0C|= (1<<UCSZ0)|(1<<UCSZ1); Veri formatının nasıl seçildiğini anlamamız için tekrar yazmaçlara bakmamız lazım. Burada UCSZ0 ve UCSZ1 bitleri bir (1) yapılmıştır. Tablodan anlaşıldığı üzere iletişim 8 bitlik bir veri genişliğinde yapılacaktır. Bundan başka iletişim UMSEL bitlerinin 00 olmasıyla asenkron olduğu için değiştirme gereği duymadık.

Şimdi veri gönderme ve veri alma fonksiyonlarını inceleyelim.

```
// veri gönderme fonksiyonu
void uart_transmit (unsigned char data)
{
    while (!( UCSRA & (1<<UDRE)));                // yazmacın boş olmasını bekle
    UDR = data;                                   // yazmaca veri yükle
}
```

Burada UDRE veri yazmacı bayrağı bir (1) olmadığı sürece program döngüde tutulur ve sonrasında UDR yazmacına fonksiyonun argüman olarak aldığı data değeri yazılır. İşlem bu kadardır.

```
// function to receive data
unsigned char uart_recieve (void)
{
    while(!(UCSRA) & (1<<RXC));                   // tüm verinin gelmesi için
bekle
    return UDR;                                   // 8 bit veriyi döndür
}
```

Öncelikle UCSRA yazmacındaki RXC bayrak bitinin bir (1) olup olmadığı kontrol edilir. Veri alma tamamlanınca bu bit bir (1) olacağı için döngüden çıkılır. Sonrasında UDR yazmacının içindeki değer fonksiyondan döndürülür.

Temel olarak USART işlemi bu kadardır. Sonraki yazıda diğer kodları inceleyeceğiz ve konumuza devam edeceğiz.

UART Kütüphanesi

avr-libc kütüphanesinde bir örnek projede yer alan usart kütüphanesi olsa da AVR'de bu tarz kütüphanelerin resmi sürümleri Arduino'da olduğu gibi yoktur. Bizim yapmamız gereken teknik veri sayfasını (datasheet) okumak ve bunun üzerinden C/C++ dilinde kod yazmaktır. Fakat AVR'nin açık kaynak programcılık anlayışında olan bir kullanıcı kitlesi olduğu için kullanıcılar tarafından yazılan kütüphaneler internette ücretsiz kullanıma

sunulmaktadır. Biz de yazdığı kütüphaneleri AVR topluluğu tarafından yaygınca kullanılan bir geliştiricinin kütüphanesini inceleyip kütüphane referansını size anlatacağız. Amatör kütüphaneler oldukça fazla olsa da kaliteli ve taşınabilir (portable) kütüphane yazmak usta işidir. Bu inceleyeceğimiz kütüphane de genel olarak taşınabilirliği ve işlevi yüksek bir kütüphanedir. Bu kütüphanenin fonksiyonlarını tek tek açıklayacağız. Öncelikle kütüphane dosyalarını aşağıdaki bağlantıdan indirip bilgisayarımızda açalım.

http://homepage.hispeed.ch/peterfleury/uartlibrary.zip

Kütüphanenin referans kılavuzu yazıldığı için kütüphaneyi açıp tek tek kodları incelemek zorunda değiliz. O yüzden öncelikle değer tanımlamaları ve makrolar ile başlayalım ve sonra fonksiyonlara geçelim.

#define UART_BAUD_SELECT(baudRate, xtalCpu) (((xtalCpu) + 8UL * (baudRate)) / (16UL * (baudRate)) -1UL)

Bu makro UART iletişim için baud oranınnı belirlemek için kullanılır. Önceki yazımızda baud oranını nasıl yazmaçlara yazmak için bir işlem uyguladığımızı anlatmıştık. Aynı işlem burada da uygulanmaktadır fakat xtalCpu ve baudRate adında iki değerden bahsedilmektedir. Bunlar bizim belirleyeceğimiz değerler olup xtalCpu sistemin saat hızı (16000000UL gibi) baudRate ise baud oranıdır (9600 gibi).

#define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate, xtalCpu) (((((xtalCpu) + 4UL * (baudRate)) / (8UL * (baudRate)) -1UL)) | 0x8000)

Bu makro yukarıdaki makronun aynısı olup Atmega'nın çift hız modunda kullandırır.

#define UART_RX_BUFFER_SIZE 32

UART iletişimde alıcının tampon belleğinin kaç bayt olacağını belirler. Bu değer 2'nin katları olmalıdır.

#define UART_TX_BUFFER_SIZE 32

UART iletişimde vericinin tampon belleğinin kaç bayt olacağını belirler. Bu değer 2'nin katları olmalıdır.

void uart_init(unsigned int baudrate)

Bu fonksiyon UART birimini hazır hale getirmeye yarar. Argüman olarak aldığı baudrate ise baud oranıdır. UART_BAUD_SELECT() makrosu ile aldığı baudrate değerini işlemci yazmaçlarına yazdırır.

unsigned int uart_getc (void)

By fonksiyon tampon bellekteki bayt verisini almaya yarar. Yani UART okuma bu fonksiyon tarafından gerçekleştirilir. Bu fonksiyon değer olarak ilk sekiz bitte tampondan okunan veriyi son sekiz bitte (üst baytta) ise durum mesajını geri döndürür. Durum mesajları şu şekildedir,

- **0** : UART biriminden veri alımı başarılır
- **UART_NO_DATA** : Alınacak bir veri yok.
- **UART_BUFFER_OVERFLOW** : Tampondan veri taşması. Yeterince hızlı veri okunmadığı için tampondaki bazı veriler kayboldu.
- **UART_OVERRUN_ERROR** : UART'da overrun durumu oluştu. UART'ın UDR yazmacındaki mevcut karakter kesme tarafından yeni karakter gelmeden önce okunamadı. Bir veya birkaç karakter kayboldu.
- **UART_FRAME_ERROR** : Veri çerçevesi hatası
  void uart_putc (unsigned char data)

Gönderilecek veri tamponuna bir baytlık veri eklemeye yarar. Aldığı argüman unsigned char yani byte değerinde olup 0-255 arasıdır.

void uart_puts (const char* s)

Gönderilecek veri tampon belleğine harf dizisi verisi koymaya yarar .

void uart_puts_p (const char* s)

Program hafızasına yerleştirilmiş harf dizisi verisini göndermeye yarar.

void uart1_init (unsigned int baudrate)

İkinci uart birimi olan ATmega mikrodenetleyiciler için ikinci UART'ı tanımlamaya yarar.

unsigned int uart1_getc (void) void uart1_putc (unsigned char data) void uart1_put2 (const char* s) void uart1_puts_p (conts char* s)

Bu fonksiyonlar UART1 için tanımlanmıştır ve yukarıda aynı addaki fonksiyonlarla aynı görevi gerine getirdiği için açıklama gereği duymuyoruz. Şimdi kütüphanenin test programını inceleyelim.

```
/*****************************************************************************
Title:    Example program for the Interrupt controlled UART library
Author:   Peter Fleury <pfleury@gmx.ch>   http://tinyurl.com/peterfleury
File:     $Id: test_uart.c,v 1.7 2015/01/31 17:46:31 peter Exp $
Software: AVR-GCC 4.x
Hardware: AVR with built-in UART/USART

DESCRIPTION:
          This example shows how to use the UART library uart.c

*****************************************************************************/
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#include "uart.h"


/* CPU saat hızını belirle değilse hata mesajı ver. */
#ifndef F_CPU
#error "F_CPU undefined, please define CPU frequency in Hz in Makefile"
#endif

/* Buradan UART baud oranını belirliyoruz */
#define UART_BAUD_RATE      9600


int main(void)
{
    unsigned int c;
    char buffer[7];
    int  num=134;


    /*
     *  Initialize UART library, pass baudrate and AVR cpu clock
     *  with the macro
     *  UART_BAUD_SELECT() (normal speed mode )
     *  or
     *  UART_BAUD_SELECT_DOUBLE_SPEED() ( double speed mode)
     */
    uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );

    /*
     * kesmeleri etkinleştiriyoruz
     */
    sei();

    /*
     *  Transmit string to UART
     *  The string is buffered by the uart library in a circular buffer
     *  and one character at a time is transmitted to the UART using interrupts.
     *  uart_puts() blocks if it can not write the whole string to the circular
     *  buffer
     */
    uart_puts("String stored in SRAM\n");

    /*
     * Transmit string from program memory to UART
```

```
     */
    uart_puts_P("String stored in FLASH\n");


    /*
     * Use standard avr-libc functions to convert numbers into string
     * before transmitting via UART
     */
    itoa( num, buffer, 10);   // convert interger into string (decimal format)
    uart_puts(buffer);        // and transmit string to UART


    /*
     * Transmit single character to UART
     */
    uart_putc('\r');

    for(;;)
    {
        /*
         * Get received character from ringbuffer
         * uart_getc() returns in the lower byte the received character and
         * in the higher byte (bitmask) the last receive error
         * UART_NO_DATA is returned when no data is available.
         *
         */
        c = uart_getc();
        if ( c & UART_NO_DATA )
        {
            /*
             * no data available from UART
             */
        }
        else
        {
            /*
             * new data available from UART
             * check for Frame or Overrun error
             */
            if ( c & UART_FRAME_ERROR )
            {
                /* Framing Error detected, i.e no stop bit detected */
                uart_puts_P("UART Frame Error: ");
            }
            if ( c & UART_OVERRUN_ERROR )
            {
                /*
                 * Overrun, a character already present in the UART UDR register
was
                 * not read by the interrupt handler before the next character
arrived,
                 * one or more received characters have been dropped
                 */
                uart_puts_P("UART Overrun Error: ");
            }
            if ( c & UART_BUFFER_OVERFLOW )
            {
                /*
                 * We are not reading the receive buffer fast enough,
                 * one or more received character have been dropped
```

```
                 */
                uart_puts_P("Buffer overflow error: ");
            }
            /*
             * send received character back
             */
            uart_putc( (unsigned char)c );
        }
    }

}
```

Burada yukarıdan itibaren başlayarak kodları Türkçe açıklayalım. Öncelikle 17. satırda uart.h başlık dosyasını programımıza ekliyoruz. Sonrasında ise F_CPU değerini tanımlamamız gerektiği için (delay kütüphanesinde olduğu gibi) bir hata mesajı ile bu denetlenmiş. Bunu da 21. ve 23. satırlar arasında görebiliriz. UART_BAUD_RATE değerini bizim tanımlamamız gerektiği için 26. satırda uart baud oranı değeri 9600 olarak tanımlanmış. 43. satırda uart_init fonksiyonunun kullanımına örnek verilmiş ve bizim önceden belirlediğimiz değerler argüman olarak alınmış. 48. satırda ilk gördüğümüz sei() fonksiyonu kullanılmıştır. Bu fonksiyon kesmeleri etkinleştirmek için kullanılır. Kütüphane UART kesmeleri ile çalıştığı için bunu açmamız gerekir. 57. satırda uart_puts() fonksiyonu ile SRAM'da tutulan string verisi gönderilmiştir. 62. satırdaki uart_puts_p() fonksiyonu ile de program hafızasında tutulan string verisi gönderilmiştir. 69. ve 70. satırlarda ise integer yani tam sayı değerinin nasıl string değerine dönüştürüleceği ve bunun gönderileceği gösterilmiştir. 76. satırda ise tek bir karakter verisinin UART ile nasıl gönderileceği gösterilmiştir.

87. satırda unsigned int olarak tanımlanan c değişkenine gelen UART verisi yüklenmiştir. 88. satırda kontrol yapıları ile c değişkeninde durum veya hata mesajları denetlenmiş ve buna göre kodların işletilmesi için örnek karar yapıları verilmiştir.

125. satırda ise alınan veriyi tekrar göndermek için bir kod yazılmıştır.

UART kütüphanesini bu örnek programdaki fonksiyonların sözdizimine ve kullanılışına bakılarak kullanmamız gereklidir. Şimdilik AVR'nin USART birimi ile ilgili anlatacaklarımızın sonuna geldik. Bir sonraki konuda görüşmek üzere.

Karakter LCD (HD44780) Kütüphanesi

Arduino'dan AVR'ye geçenlerin en büyük sıkıntısı artık kütüphane olmayışı ve çoğu işi sıfırdan kendimiz yapmamız gerektiği algısıdır. Temel giriş ve çıkış bir şekilde halledilse de geri kalan işimizi kolaylaştıran kütüphaneler elimizin altında yoktur. Fakat ufak bir araştırma yapıldığı zaman aslında AVR için oldukça fazla kütüphane yazıldığının farkına varabilirsiniz. Arduino AVR kütüphanelerinin yazılıp paylaşılmasına bir bakıma engel olmaktadır. Çünkü AVR programcısı olacak kişiler Arduino'yu tercih etmektedir. O yüzden AVR bilenler AVR kütüphanesi yazmak yerine Arduino kütüphanesi yazmayı daha yararlı bulmaktadır. Kütüphane bulmada ilk engelimiz binlerce Arduino kütüphanesi olup AVR kütüphanelerinin arka planda kalmasıdır. Yine de Github'dan arattığımız zaman yeterli miktarda kütüphane bulabiliyoruz. Mevcut olmayan kütüphaneleri ise Arduino kütüphanelerini değiştirerek yazmak mümkündür. Şimdi karakter LCD kullanımı için yazılmış bir kütüphanenin bağlantısını ve referansını açıklayacağız.

Önümüzde yazılacak onlarca ders olduğu için uygulama yapmaya vakit bulamıyoruz. Uygulama eksiği olanlar internette yer alan onlarca Arduino uygulamasına bakabilir. Bu dersler Arduino ile yeterli uygulamayı yapmış, nasıl devre kuracağını anlamış kişiler için yazılmıştır. O yüzden bu konuda eksiği olanlar şimdilik Arduino uygulamalarına bakabilir.

Karakter LCD kütüphanesi aşağıdaki bağlantıdan indirilebilir.
http://homepage.hispeed.ch/peterfleury/lcdlibrary.zip

Tanımlar

#define LCD_CONTROLLER_KS0073 0 Bu tanım LCD üzerinde bulunan kontrolcü entegreyi seçmeye yarar. HD44780 kullanılıyorsa sıfır (0), KS0073 kullanılıyorsa bir (1) yapılmalıdır. Günümüzde genelde HD44780 entegreli LCD ekranlar kullanılmaktadır. Buna dikkat edilmezse ekran çalışmayabilir.

Bu tanımlamalar lcd_definations.h dosyasında bulunmaktadır. Program yazmadan önce muhakkak kullanacağınız ekranın değerlerini bu tanımlardan değiştirmeniz gereklidir. Eğer değiştirmezseniz programınız çalışmayabilir.

LCD EKRAN AYARLARI

#define LCD_LINES 2 Bu tanım LCD ekranın kaç satır olacağını belirler. Genellikle 2 veya 4 satır olmaktadır. Buradan kaç satır olacağını belirleyebilirsiniz.

#define LCD_DISP_LENGTH 16

Bu tanım LCD ekranının kaç sütün olacağını belirler. Eğer elimizde 16×2 LCD varsa burası 16 olmalıdır. Eğer 20×4 gibi bir ekran varsa burayı 20 yapmalıyız.

#define LCD_LINE_LENGTH 0x40

Bu tanım LCD'nin satırındaki veri uzunluğunu belirlemeye yarar.

#define LCD_START_LINE1 0x00

LCD'nin birinci satırındaki DRAM adresini belirlemeye yarar.

#define LCD_START_LINE2 0x40

LCD'nin ikinci satırındaki DRAM adresini belirlemeye yarar.

#define LCD_START_LINE3 0x14

LCD'nin üçüncü satırındaki DRAM adresini belirlemeye yarar.

#define LCD_START_LINE4 0x54

LCD’nin dördüncü satırındaki DRAM adresini belirlemeye yarar.

#define LCD_WRAP_LINES 0

LCD’de satırları döndürmeye yarar. Görünen satırın sonunu döndürür.

LCD AYAK AYARLARI

#define LCD_IO_MODE 1 0: Hafıza haritalantırma modu, 1: IO Port Modu

#define LCD_PORT PORTA

LCD ayakları için port seçimi

#define LCD_DATA0_PORT LCD_PORT

4-bit verinin 0. biti için port seçimi

#define LCD_DATA1_PORT LCD_PORT

4-bit verinin 1. biti için port seçimi

#define LCD_DATA2_PORT LCD_PORT

4-bit verinin 2. biti için port seçimi

#define LCD_DATA3_PORT LCD_PORT

4-bit verinin 3. biti için port seçimi

#define LCD_DATA0_PIN 0

4-bit verinin 0. biti için ayak seçimi

#define LCD_DATA1_PIN 1

4-bit verinin 1 . biti için ayak seçimi

#define LCD_DATA2_PIN 2

4-bit verinin 2. biti için ayak seçimi

#define LCD_DATA3_PIN 3

4-bit verinin 3. biti için ayak seçimi

#define LCD_RS_PORT LCD_PORT

RS ayağı için port seçimi. Buraya başka port adı da girilebilir.

#define LCD_RS_PIN 4

RS ayağı için ayak seçimi.

#define LCD_RW_PORT LCD_PORT

RW ayağı için port seçimi.

#define LCD_RW_PIN 5

RW ayağı için pin seçimi

#define LCD_E_PORT LCD_PORT

Enable ayağı için port seçimi

#define LCD_E_PIN 6

Enable ayağı için ayak seçimi

#define LCD_DELAY_BOOTUP 16000

Açılışta kaç mikrosaniye bekleneceği belirlenir.

#define LCD_DELAY_INIT 5000

Yükleme komutu gönderildikten sonra kaç mikrosaniye bekleneceği belirlenir.

#define LCD_DELAY_INIT_REP 64

Yükleme komutu tekrarlandıktan sonra kaç mikrosaniye bekleneceği belirlenir.

#define LCD_DELAY_INIT_4BIT 64

4-bit modunu seçtikten sonraki beklemeyi belirler. (mikrosaniye)

#define LCD_DELAY_BUSY_FLAG 4

Meşgul bayrağı temizlenip adres satırı güncellendikten sonraki mikrosaniye beklemenin ne kadar olacağını kararlaştırır.

#define LCD_DELAY_ENABLE_PULSE 1

Enable ayağının sinyal genişliğini mikrosaniye olarak ayarlar.

LCD KOMUT AÇIKLAMALARI

Bu komutlar lcd_command() fonksiyonu ile kullanılabilir.

```
#define LCD_CLR 0 /* DB0: clear display */
#define LCD_HOME 1 /* DB1: return to home position */
#define LCD_ENTRY_MODE 2 /* DB2: set entry mode */
#define LCD_ENTRY_INC 1 /* DB1: 1=increment, 0=decrement */
#define LCD_ENTRY_SHIFT 0 /* DB2: 1=display shift on */
#define LCD_ON 3 /* DB3: turn lcd/cursor on */
#define LCD_ON_DISPLAY 2 /* DB2: turn display on */
#define LCD_ON_CURSOR 1 /* DB1: turn cursor on */
#define LCD_ON_BLINK 0 /* DB0: blinking cursor ? */
#define LCD_MOVE 4 /* DB4: move cursor/display */
#define LCD_MOVE_DISP 3 /* DB3: move display (0-> cursor) ? */
#define LCD_MOVE_RIGHT 2 /* DB2: move right (0-> left) ? */
#define LCD_FUNCTION 5 /* DB5: function set */
#define LCD_FUNCTION_8BIT 4 /* DB4: set 8BIT mode (0->4BIT mode) */
#define LCD_FUNCTION_2LINES 3 /* DB3: two lines (0->one line) */
#define LCD_FUNCTION_10DOTS 2 /* DB2: 5x10 font (0->5x7 font) */
#define LCD_CGRAM 6 /* DB6: set CG RAM address */
#define LCD_DDRAM 7 /* DB7: set DD RAM address */
#define LCD_BUSY 7 /* DB7: LCD is busy */
#define LCD_ENTRY_DEC 0x04 /* display shift off, dec cursor move dir */
#define LCD_ENTRY_DEC_SHIFT 0x05 /* display shift on, dec cursor move dir */
#define LCD_ENTRY_INC_ 0x06 /* display shift off, inc cursor move dir */
#define LCD_ENTRY_INC_SHIFT 0x07 /* display shift on, inc cursor move dir */
#define LCD_DISP_OFF 0x08 /* display off */
#define LCD_DISP_ON 0x0C /* display on, cursor off */
#define LCD_DISP_ON_BLINK 0x0D /* display on, cursor off, blink char */
#define LCD_DISP_ON_CURSOR 0x0E /* display on, cursor on */
#define LCD_DISP_ON_CURSOR_BLINK 0x0F /* display on, cursor on, blink char */
#define LCD_MOVE_CURSOR_LEFT 0x10 /* move cursor left (decrement) */
#define LCD_MOVE_CURSOR_RIGHT 0x14 /* move cursor right (increment) */
#define LCD_MOVE_DISP_LEFT 0x18 /* shift display left */
#define LCD_MOVE_DISP_RIGHT 0x1C /* shift display right */
#define LCD_FUNCTION_4BIT_1LINE 0x20 /* 4-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_4BIT_2LINES 0x28 /* 4-bit interface, dual line, 5x7 dots */
#define LCD_FUNCTION_8BIT_1LINE 0x30 /* 8-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_8BIT_2LINES 0x38 /* 8-bit interface, dual line, 5x7 dots */
#define LCD_MODE_DEFAULT ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC) )
```

LCD Fonksiyonları

void lcd_init(uint8_t dispAttr)

Ekranı tanımlar ve imleci belirler.

Parametreler dispAttr: – **LCD_DISP_OFF** : Ekran Kapalı – **LCD_DISP_ON** : Ekran Açık, İmleç Kapalı – **LCD_DISP_ON_CURSOR** : Ekran açık, imleç açık – **LCD_DISP_ON_CURSOR_BLINK** : Ekran açık, Yanan imleç

Örneğin ekranımız açık olsun ve imlecimiz sabit olsun bunun için şöyle bir kod yazacağız, **lcd_init(LCD_DISP_ON_CURSOR);**

void lcd_clrscr (void)

Ekranı temizler ve imleci başa getirir. Örnek kullanım:

lcd_clrscr();

void lcd_home (void)

İmleci başa alır. Örnek kullanım:

lcd_home();

void lcd_gotoxy (uint8_t x, uint8_t y)

İmleci belirlenen bir konuma getirir.

Parametreler

x: Yatay konum (0 en sol) **y**: Dikey konum (0 en üst)

Örnek bir kullanım şöyle olabilir, **lcd_gotoxy(5,1);**

void lcd_putc (char c)

Mevcut imleç konumunda bir karakter gösterir

Parametreler c : gösterilecek karakter

Örnek bir kullanım şöyle olabilir, **lcd_putc (‘a’);**

void lcd_puts (const char* s)

Otomatik satır beslemesi ile harf dizisini gösterir. **Parametreler s** : Gösterilecek harf dizisi verisi Örnek bir kullanım şöyle olabilir, **lcd_puts(“Merhaba”);**

void lcd_puts_p (const char* progmem_s)

Program hafızasındaki harf dizisi verisini gösterir. **Parametreler**

progmem_s Program hafızasındaki gösterilecek harf dizisi.

void lcd_command (uint8_t cmd)

LCD'ye işlemci komutu gönderilir.

Parametreler cmd : LCD gönderilecek işlemci komutu. HD44780 teknik veri kitapçığında bulunabilir.

void lcd_data (uint8_t data)

LCD'ye bayt verisi gönderir. Parametreler **data** : LCD sürücüye gönderilecek bayt verisi. Hd44780 teknik veri kitapçığına bakın.

LCD'ye Integer Yazdırmak

Yukarıdan anlaşılacağı üzere LCD'ye sadece karakter ve string yazdırıldığını fark etmişsinizdir. Şimdi sprintf() fonksiyonu ile integer değerini harf dizisine dönüştürüyoruz ve LCD'ye yazdırıyoruz. Aynı zamanda LCD ile örnek bir programı da görebilirsiniz.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <stdio.h>
#include "lcd.h"

int main (void)
{
        lcd_init(LCD_DISP_ON);
        lcd_clrscr();
        lcd_home();
        char str [16];
        int pi = 30;
        sprintf(str, "Pi = %i", pi);
        lcd_puts(str);
        while(1)
        {
```

LCD'nin çalışma prensibini Gömülü Sistemler kategorisinde, LCD ile örnek uygulamaları ise AVR uygulamalarına başladığımız zaman yazacağız. Şimdilik ihtiyacı olanlar için kütüphaneyi açıklayarak derslerimize devam edelim. Şimdiye kadar anlattığımız bilgilerle bu kütüphaneyi kullanmak mümkün olduğu için bu kadarıyla yetinip bir sonraki konuya geçelim.

Zamanlayıcı ve Sayıcılara Giriş (Timer/Counter)

Artık büyük konuları anlatmanın zamanı geldiği için şimdiye kadar anlattıklarımızı tekrar bir gözden geçirelim. Şimdiye kadar anlattıklarımız konuların yarısını bile olmasa dahi size bu konulardan daha büyük bir şey kazandırmış olmalıdır. Çünkü hiçbir zaman üst seviyeden sığ bir şekilde konuları geçmedik ve en gerekli yerleri üşenmeden anlattık. Yeri geldi kütüphanelerin fonksiyonlarını tek tek açıkladık yeri geldi bir kütüphanenin nasıl yazıldığını anlatmak için kodlarını tek tek açıkladık. Çoğu zaman teknik veri kitapçığındaki bilgileri buraya getirdik ve hepsini anlatmaya çalıştık. Bu dersler yazılırken kısıtlı zamana sahip olduğum için imla, yazım hatası ve dikkatten kaçan noktalar bulunabilir. Bunu dile getirmenizde bir sakınca yoktur. Çünkü dersi yazar yazmaz yayınlayıp bir an önce sonraki derse geçiyorum. Bunun sebebi bütün kısıtlı zamanımı ders yazmaya ayırmak istememden dolayıdır. İlk dersi yazmaya başlamamızdan itibaren 10 gün geçmiş ve bu noktaya kadar toplamda 24 ders yazdık.

Şimdi işlediğimiz konuları madde madde gözden geçirelim,

- AVR mikrodenetleyicisini tanıttık ve gerekli derleyici, yazılım ve programcılardan bahsettik.
- Temel giriş ve çıkış özelliklerinin tamamını açıkladık.
- ADC birimini açıkladık
- USART birimini açıkladık
- LCD kütüphanesini açıkladık

Diğer konularla kıyaslayınca yolun daha başında olduğumuzu görüyoruz. Bizi yolun başında hissettiren konular ise Zamanlayıcılar ve Kesmelerden başkası değil. Bu konuları atladıktan sonra çeşitli iletişim, güç ve çevre birimi konuları kalıyor. Bunlar zamanlayıcılar ve kesmeler kadar mikrodenetleyicinin temel özellikleri içerisinde değildir.

Derslerin nasıl bir formatta olacağına başta hiç karar vermeyip kafama estiği gibi yazmaya başlamıştım. Böylece dersin formatı kendiliğinden ortaya çıktı. Derslerin formatına göre her konuyu şu sırada inceleyeceğiz.

- Anlatılacak konuya temel bir giriş yapılır.
- Anlatılacak konu teknik veri kitapçığından anlatılır.
- Anlatılacak konu hakkında örnek kod varsa açıklanır.
- Anlatılacak konu hakkında kütüphane varsa anlatılır.

Bu dersler sadece AVR dersi olduğu için C dili, elektronik, devre kurma vs. hakkında fazla bir şey anlatmayacağız. AVR dışında modülleri, entegreleri derse dahil etmeyeceğiz ve AVR'yi de mümkün olduğu kadar C dili için alt seviyede anlatacağız.

Şimdi zamanlayıcı ve sayıcılara giriş yapalım.

Zamanlayıcılar sadece mikrodenetleyicilerin dünyasında değil günlük hayatımızda da sıkça kullanılır. Hatta saat ve kronometrenin olması da şart değildir. İşlerimizi sabah, öğle, ikindi, kuşluk, akşam, gece, seher gibi pek çok vakte göre tayin ederiz. Mikrodenetleyiciler de zamana bağlı uygulamaları yerine getirmek için işlemci saatini kullanabilseler de bu

oldukça verimsiz olacaktır. Çünkü işlemciyi diğer bütün işlemler için kullanırken aynı zamanda da zamanı saymakla meşgul etmek zorunda kalırız. Bunun için işlemciden ayrı olarak zamanlayıcı birimleri yer almakta ve işlemci bu birimleri gerektiğinde okuyarak işlerini bunlara göre yapmaktadır. Hatta istenildiğinde kesmeler de kullanılarak mikrodenetleyici hiç bununla meşgul olmadan sadece işle meşgul olabilmektedir. Birkaç mikrosaniyeden birkaç saate kadar uzayan bu zaman dilimi mikrodenetleyicinin ihtiyacı olan zamanlama işlemini fazlasıyla karşılamaktadır. Aynı zamanda bu birimlerin yüksek doğrulukta olması işleri güvenle bu birimlere teslim edebilmemizi sağlar.

Zamanlayıcıların temeli yazmaçlardır. Bu yazmaçlardaki veri sırayla artar ve bu yazmaçlar okunarak zamana bağlı işlemler yapılır. Mikrodenetleyicinin özelliklerini anlatırken iki ayrı tipte zamanlayıcı olduğundan bahsetmiştik. 8-bit ve 16-bit olmak üzere ikiye ayrılan bu zamanlayıcılardan 8-bit olanı 0-255 arası, 16-bit olanı ise 0-65535 arası değeri içinde saklayabilir. Ayrıca zamanlayıcılar sayıcı olarak kullanılabilse de bu sayıcı özelliği içeride farklı bir özellik olmayıp saat frekansını dışarıdan alıp ona göre değeri artırmakla olur. Zamanlayıcıların yazmaçları azami değere geldiği zaman taşar ve tekrar sıfırlanır. Atmega328P mikrodenetleyicisinde TC0, TC1 ve TC2 olmak üzere üç adet ayrı zamanlayıcı bulunur. Bunlardan ikisi (TC0 ve TC2) 8-bit olup biri (TC1) 16 bittir. Zamanlayıcının en önemli özelliği yukarıda bahsettiğimiz gibi işlemciden bağımsız olmasıdır. Eğer işlemciye bağımlı bir zamanlayıcı olsaydı doğrulukta büyük sıkıntı çekilirdi. Çünkü işlemci başka kodları işlemekle meşgul olmaktadır ve aynı anda iki kodu işleyememektedir. Atmega328P'de bulunan 3 zamanlayıcının özelliklerini verelim ve diğer konuları sonraki derse bırakalım.

TC0 – 8-bit Zamanlayıcı/Sayıcı (PWM destekli)

- İki bağımsız çıkış karşılaştırma ünitesi
- Çift tamponlu çıkış karşılaştırma yazmacı
- Karşılaştırma eşleşmesinde zamanlayıcıyı sıfırlama
- PWM (Pulse Width Modulator)
- Ayarlanabilir PWM Periyodu
- Frekans üreteci
- Üç adet bağımsız kesme kaynağı (TOV0, OCF0A ve OCF0B)

TC1 – 16-bit Zamanlayıcı/Sayıcı (PWM destekli)

- Gerçek 16-bit tasarım (16-bit PWM'yi destekler.)
- İki adet bağımsız çıkış karşılaştırma ünitesi
- Çift tamponlu çıkış karşılaştırma yazmacı
- Bir adet giriş yakalama ünitesi
- Giriş yakalaması gürültü önleyici
- Karşılaştırma eşleşmesinde zamanlayıcıyı sıfırlama
- PWM desteği
- Ayarlanabilir PWM periyodu
- Frekans üreteci
- Harici olay sayıcı
- Bağımsız kesme kaynakları (TOV, OCFA, OCFB ve ICF)

TC2 – 8-bit Zamanlayıcı/Sayıcı PWM ve Asenkron Çalışma Destekli

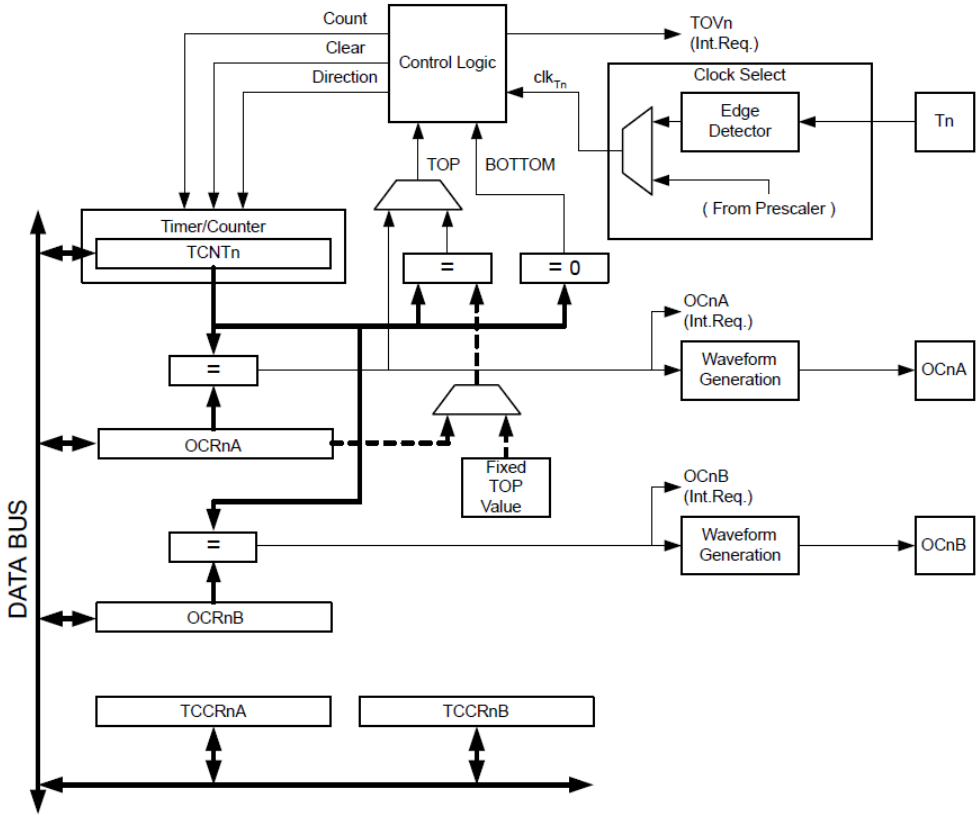
- Kanal Sayıcı
- Karşılaştırma eşleşmesinde zamanlayıcıyı sıfırlama
- PWM
- Frekans üreteci
- 10-bit saat ön derecelendiricisi
- Taşma ve eşleşme kesme kaynakları (TOV2, OCF2A ve OCF2B)
- Harici saat kristali bağlanabilir (32kHz)

TC0 Zamanlayıcısı

Zamanlayıcılar oldukça uzun bir konu olduğu için bizi epey meşgul edecektir. Öncelikle zamanlayıcıları teknik veri sayfasından özetle tanıyıp sonrasında yazmaçları öğrenerek işe başlayalım. Elimizde tek bir zamanlayıcı yerine üç ayrı zamanlayıcı bulunduğu için bunları tek tek sırayla ele almamız gereklidir. Teknik veri sayfasında da bu üç zamanlayıcı ayrı başlıklar altında uzunca açıklanmıştır. Oldukça uzun sürecek konumuza TC0 zamanlayıcısını anlatmakla başlayalım.

Önceki yazımızda özelliklerini verdiğimiz TC0 modülü genel amaçlı 8-bit zamanlayıcı/sayıcı modülüdür. Ayrıca bağımsız karşılaştırma üniteleri ve PWM desteği mevcuttur. Basitleştirilmiş blok diyagramında görüleceği üzere CPU erişimli giriş ve çıkış yazmaçları, giriş ve çıkış bitleri, giriş ve çıkış ayakları koyu renkte gösterilmiştir. Aygıtın yazmaçları ve bit özelliklerine ise yazmaçları açıkladığımız sırada geleceğiz. TC0 modülünün blok diyagramı aşağıdaki gibidir.

Figure 19-1. 8-bit Timer/Counter Block Diagram



Resim: ATmega328P – Microchip Technology ,sf. 126,http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 25.08.2018

Şema üzerinden incelediğimizde kabaca bir adet kontrol birimi, saat birimi ve zamanlayıcı biriminden oluştuğunu görüyoruz. Ayrıca pek çok yazmacın yer aldığını söylememiz mümkündür. Bu yazmaçların fazla olmasının yanında üç adet ayak çıkışı da vardır. Bunlar biri giriş olup Tn olarak isimlendirilmiştir. İki adet ise çıkış ayağı yer alıp dalga üretecine bağlıdır. Timer/Counter biriminin içinde yer alan TCNT yazmacı kontrol yapısına bağlı olduğu gibi veri yoluna da bağlıdır. Buradan bunun zamanlayıcı değerini barındıran yazmaç olduğunu söylememiz mümkündür. Ayrıca alt tarafta yine ICPn olarak bir adet giriş ayağı daha bulunmaktadır. Şemayı yazmaçları anladıktan sonra tekrar gözden geçirdiğimizde çok daha anlaşılır olacaktır. Şimdi konumuza devam edelim.

Zamanlayıcıları anlatırken kullanılan üç tabir vardır. Bunlar alt (bottom), azami (max) ve üst (top) olarak isimlendirilir. Bunların anlamını sırasıyla açıklayalım.

BOTTOM: Sayıcı sıfır olduğunda BOTTOM noktasına varır.

MAX : Sayıcı 0xFF olduğunda azami değerine ulaşır. (16-bit için 0xffff)

TOP : Sayıcı sayım sırasında kendi en yüksek değerine ulaştığında TOP noktasına ulaşır. TOP değeri MAX değerine veya OCR1A yazmacındaki değere sabitlenebilir.

TC0 modülü ile ilgili geri kalan bilgiler yazmaçlar üzerinden anlatıldığı için öncelikle yazmaçları anlatalım.

TCCR0A – TC0 Denetim Yazmacı

Bit	7	6	5	4	3	2	1	0
	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Bit 7:6 – COM0An : A kanalı için karşılaştırma çıkış modu [n = 1:0]

Bu bitler karşılaştırma çıkışı ayağının (OC0A) davranışını belirler. Eğer COM0A bitlerinden biri bir (1) yapılırsa OC0A çıkışı normal port fonksiyonunu geçersiz kılar. Çıkış sürücüsünü etkinleştirmek için veri yönü yazmacı (DDR) ‘da OC0A ya karşılık gelen ayak bir (1) yapılmak zorundadır. OC0A bir ayağa bağlandığında COM0A[1:0] bitleri WGM0[2:0] bit ayarına bağımlıdır. Aşağıdaki tabloda WGM0[2:0] bitlerinin normal ya da CTC moduna alındığındaki COM0A bitlerinin durumunu görüyoruz.

Karşılaştırma çıkış modu (PWMsiz)

COM0A1	COM0A0	Açıklama
0	0	Normal Port Çalışması, OC0A bağlı değil.
0	1	Karşılaştırma Eşleşmesinde OC0A'yı Aç/Kapa
1	0	Karşılaştırma Eşleşmesinde OC0A'yı Temizle
1	1	Karşılaştırma Eşleşmesinde OC0A'yı Aç

Aşağıdaki tablo WGM0[1:0] bitleri hızlı PWM moduna alındığındaki COM0A [1:0] bitlerinin fonksiyonunu açıklar.

COM0A1	COM0A0	Açıklama
0	0	Normal Port Çalışması, OC0A bağlı değil.
0	1	WGM02 = 0 : Normal port çalışması, OC0A bağlı değil. WGM02 = 1 OC0A'yı karşılaştırma eşleşmesinde Aç/Kapa
1	0	Karşılaştırma eşleşmesinde OC0a'yı temizle ve OC0A'yı BOTTOM'da aç. (terslememe modu)
1	1	OC0A'yı karşılaştırma eşleşmesinde aç, OC0A'yı BOTTOM'da temizle.

Aşağıdaki tablo WGM0[2:0]bitlerinin faz düzeltmeli PWM modundayken COM0A[1:0]bitlerinin işleyişini gösterir.

COM0A1	COM0A0	Açıklama
0	0	Normal Port işleyişi, OC0A bağlı değil.

0	1	WGM02 = 0 : Normal Port işleyişi, OC0A bağlı değil. WGM02 = 1 : OC0A'yı karşılaştırma eşleşmesine Aç/Kapa
1	0	OC0A'yı karşılaştırma eşleşmesinde yukarı sayarken temizle ve OC0A'yı BOTTOM'a ayarla.
1	1	OC0A'yı karşılaştırma eşleşmesinde yukarı sayarken ayarla ve OC0A yı aşağı sayarken temizle.

Bit 5:4 – COM0Bn : Kanal için karşılaştırma çıkış modu B [n = 1:0]

Bu bitler çıkış karşılaştırma ayağının (OC0B) davranışını belirler. Eğer bir yada iki COM0B[1:0] bitleri bir (1) yapılırsa OCB0B o ayağın port işleyişini geçersiz kılar. OC0B ayağa bağlandığında COM0B[1:0] fonksiyonu WGM0[2:0] bitine bağlı halde çalışır. Aşağıdaki tablo WGM0[2:0]bitlerinin normal CTC modundayken COM0B[1:0] bitlerinin fonksiyonunu vermiştir.

COM0B1	COM0B0	Açıklama
0	0	Normal Port Çalışması, OC0B bağlı değil.
0	1	Karşılaştırma Eşleşmesinde OC0B'yı Aç/Kapa
1	0	Karşılaştırma Eşleşmesinde OC0B'yı Temizle
1	1	Karşılaştırma Eşleşmesinde OC0B'yı Aç

Aşağıdaki tablo WGM0[1:0] bitleri hızlı PWM moduna alındığındaki COM0B [1:0] bitlerinin fonksiyonunu açıklar.

COM0B1	COM0B0	Açıklama
0	0	Normal Port Çalışması, OC0A bağlı değil.
0	1	Rezerve

1	0	Karşılaştırma eşleşmesinde OC0a'yı temizle ve OC0A'yı BOTTOM'da aç. (terslememe modu)
1	1	OC0A'yı karşılaştırma eşleşmesinde aç, OC0A'yı BOTTOM'da temizle.

Aşağıdaki tablo WGM0[2:0]bitlerinin faz düzeltmeli PWM modundayken COM0B[1:0]bitlerinin işleyişini gösterir.

COM0A1	COM0A0	Açıklama
0	0	Normal Port işleyişi, OC0A bağlı değil.
0	1	Rezerve
1	0	OC0B'yi karşılaştırma eşleşmesinde yukarı sayarken temizle ve OC0B'yi BOTTOM'a ayarla.
1	1	OC0B'yi karşılaştırma eşleşmesinde yukarı sayarken ayarla ve OC0B'yi aşağı sayarken temizle.

Bit 1:0 – WGM0n : Dalga üreteci modu [n = 1:0]

TCCR0B yazmacındaki WGM02 bitiyle beraber kullanıldığında bu bitler sayıcının sayma sırasını denetler, azami (TOP) değerin kaynağını ve hangi tip dalga formunun kullanılacağını belirler. Çalışma modları olarak Normal, Karşılaşma eşleşmesinde zamanlayıcıyı temizleme (CTC) ve iki tip PWM modu kullanılabilir. Dalga formu üreteci modunun bit açıklaması aşağıdaki tablodaki gibidir.

Table 19-9. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Resim: ATmega328P – Microchip Technology , sf. 140,http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 25.08.2018

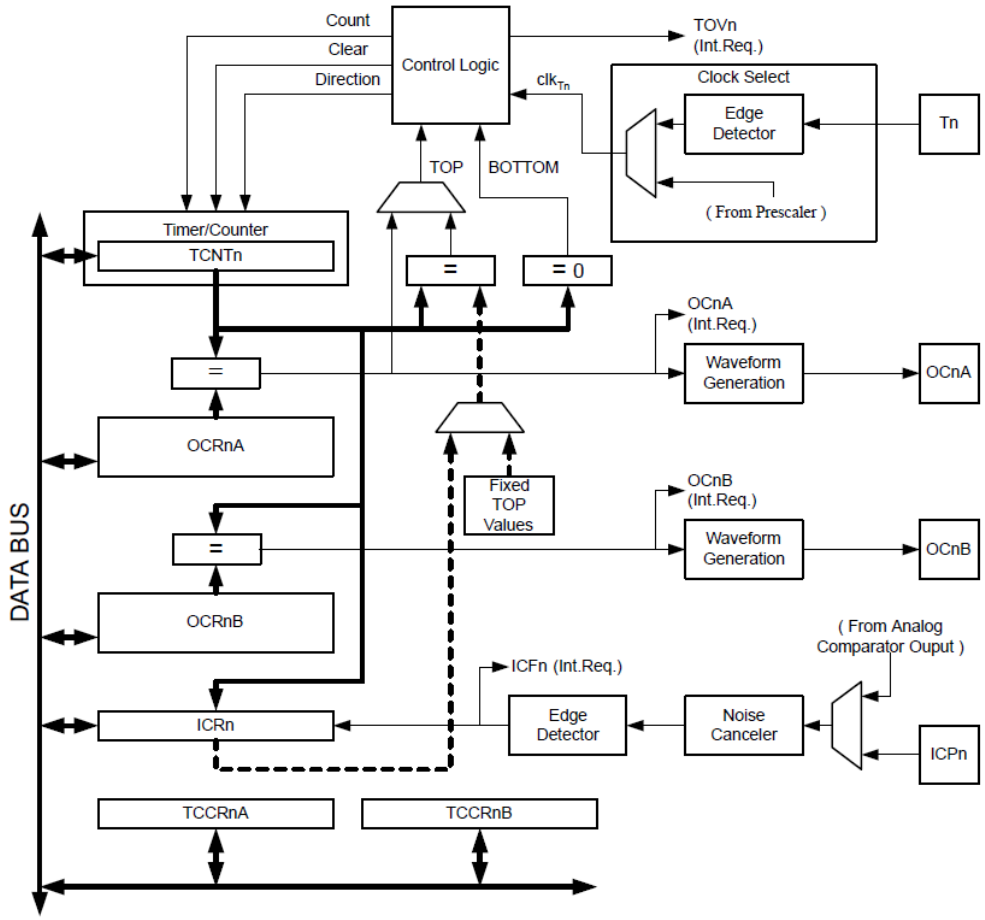
Şimdilik TC0 modülünün sadece ilk yazmacını anlattık. Geriye anlatılacak pek çok başlık olduğu gibi daha 2 adet zamanlayıcı modülü daha var. Şimdilik dersi burada bitirelim ve sonraki derse geçelim

TC1 Zamanlayıcısı

TC0 zamanlayıcısını nihayet bitirsek de geriye TC1 ve TC2 zamanlayıcıları kaldı. Bundan sonra ise bu zamanlayıcıların kullanımını ve örnek kodları inceleyeceğiz. TC1 zamanlayıcısı TC0 zamanlayıcısına benzese de diğerinden en önemli farkı bu zamanlayıcının 16-bit olmasıdır. Böylelikle daha geniş değer aralığına sahip olan sayma değeri daha esnek bir zamanlama imkanı sunar. Bu esneklik ön derecelendirici değerini artırarak bir noktaya kadar artırılsa da 8 bitlik bir değer ile oldukça az oluyordu. 16 bitlik zamanlayıcıda bu geniş değer aralığı ve ön derecelendiriciler ile oldukça uzun süre aralıklarında çalışan bir zamanlayıcı elde edebiliriz.

TC1 zamanlayıcısının yazmaçlarını incelediğimde TC0 zamanlayıcısının yazmaçları ile hemen hemen aynı olduğunu görürüz. Aradaki en büyük fark sayaç ve karşılaştırma değerlerini barındıran yazmaçların her birinin toplamda 16 bitlik değer bulundurması olarak görünüyor. Yine bu değerler 8 bitlik yüksek ve düşük bayt değeri olarak ayrılsa da ikisinin toplamı 16 bit olduğu için toplamda 16 bitlik değer saklayabiliyor. Yine TC0 yazmaçlarında olduğu gibi ayar bitleri, kesme bitleri, bayrak bitleri bu yazmaçlarda da mevcuttur.

Doğrudan yazmaçlardan bahsederek önceki konuyu tekrarlamamak için şimdi TC0'ı anlatırken değinmediğimiz konulardan bahsederek dersimizi devam ettireceğiz. Yine yazmaç kısmında belli noktalara değineceğiz ve iki zamanlayıcının yazmaçlarını karşılaştırarak farklı noktaları dersimize dahil edeceğiz. Öncelikle anlatacağımız konunun anlaşılması için TC1'in blok diyagramını verelim.



Resim : Resim: ATmega328P – Microchip Technology ,sf. 150,http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 25.08.2018

Burada fark edildiği üzere TC1 zamanlayıcısının yapısı TC0 zamanlayıcısı ile hemen hemen aynıdır. O yüzden zamanlayıcının yapısını açıklamamız diğer zamanlayıcıları tanıma konusunda size yardımcı olacaktır. Bundan önce 16 bitlik yazmaçları kullanma konusunda biraz bilgi vermemiz gereklidir.

16 Bit Yazmaçlara Erişim

TCNT1, OCR1A/B ve ICR1 yazmaçları 16 bit olup AVR işlemcisinin 8 bitlik veri yolu ile erişilebilir. Ancak bu erişim bayt tabanlı olup iki okuma ve yazma işlemi ile sağlanır. Her 16-bit zamanlayıcı tek bir 8-bit geçici yazmaca sahip olup 16 bitlik verinin yüksek baytını içinde bulundurur. Aynı geçici yazmaç bütün 16 bitlik yazmaçlar ile paylaştırılır. Düşük bayta erişim 16 bitlik okuma ya da yazma işlemini tetikler. Eğer 16 bitlik yazmacın düşük baytı işlemci tarafından yazılırsa yüksek baytı TEMP yazmacında saklanır ve düşük bayt ile beraber 16 bitlik yazmaca aynı saat çeviriminde kopyalanır. 16 bitlik yazmacın düşük baytı okunduğunda yüksek bayt TEMP yazmacına işlemci tarafından kopyalanır.

16 bitlik yazmaca erişim Assembly ve C dillerinde farklı şekilde yürür. Biz şimdilik C dilinde verilmiş örnek kodu buraya koyacağız.

```
unsigned int i;
...
/* TCNT1 yazmacına bir değer ver */
TCNT1 = 0x1FF;
/* TCNT1 yazmacını oku*/
i = TCNT1;
...
```

Burada i değişkeninin unsigned int cinsinde olduğunu unutmayalım. Bu değişken 0 ile 65535 arasındaki 16 bitlik değeri saklayabilir.

Eğer TCNT1 (Önceki konuda bahsettiğimiz üzere sayıcı değerini bulunduran yazmaç.) yazmacını okumak veya yazmak bir kesmeye sebep oluyorsa öncelikle kesmeleri kapatıp sonra okuma ve yazma işlemini yapıp bunun ardından tekrar eski mikrodenetleyici ayarını yapmak gereklidir. Okuma yapmak için örnek fonksiyon aşağıdaki gibi olmalıdır.

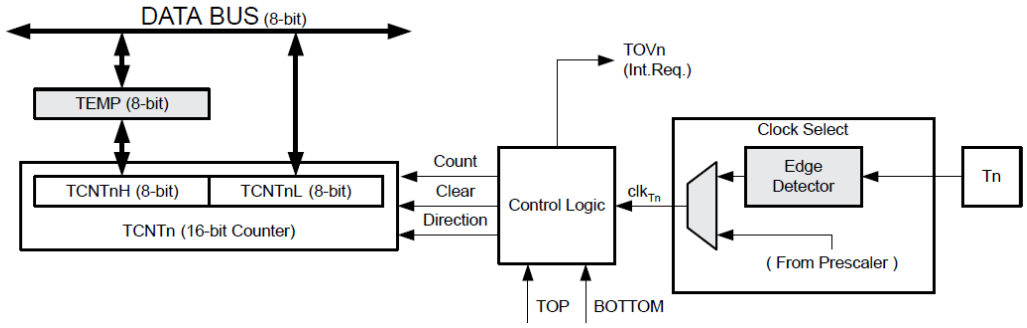
```
unsigned int TIM16_ReadTCNT1( void )
{
unsigned char sreg;
unsigned int i;
/* Genel kesme bayrağını kaydet.  */
sreg = SREG;
/*Kesmeleri kapat */
_CLI();
/* TCNT1 değerini oku */
i = TCNT1;
/* Eski değeri yükle */
SREG = sreg;
return i;
}
```

Yazma işlemindeki fonksiyonda ise sadece TCNT1 yazmacı ve i değişkeninin yerleri değişmelidir.

Sayıcı Ünitesi

16 bitlik zamanlayıcı ve sayıcı ünitesinin ana parçası programlanabilir 16-bit iki yönlü sayıcı ünitesidir. Aşağıda blok diyagramını görebilirsiniz.

Figure 20-2. Counter Unit Block Diagram



Burada bazı sinyalleri açıklamak gereklidir. Bu sinyallerin açıklaması aşağıdaki tabloda verilmiştir.

Sinyal Adı	Açıklama
Count	TCNT1 yazmacının birer azaltımı veya artırımı
Direction	Azaltma ve Artırım seçimi
Clear	TCNT1 yazmacını temizleme (Bitlerin sıfırlanması)
clkt1	Zamanlayıcı saat sinyali
TOP	TCNT1 azami değerine ulaştığında verilecek sinyal
BOTTOM	TCNT1 dip değerine ulaştığında verilecek sinyal (0)

16-bit sayıcı iki adet 8-bit giriş ve çıkış hafıza konumuna haritalandırılmıştır. Yüksek Sayaç (TCNT1H) sayacın üst sekiz bitini barındırırken, Alçak Sayaç (TCNT1L) sayacın alt sekiz bitini barındırır. İşlemci TCNT1H yazmacının giriş ve çıkış konumuna eriştiğinde işlemci geçici yüksek bit yazmacına (TEMP) erişmiş olur. TCNT1L okunduğunda geçici yazmaç güncellenir. Ayrıca TCNT1L yazmacına veri yazıldığı zaman TCNT1H yazmacı güncellenir. Böylelikle bu özellik işlemcinin tek bir saat çeviriminde 8 bitlik veri yolu üzerinden 16 bit veri okumasını sağlar.

Çalışma moduna bağlı olarak sayıcı her saat çevriminde (clkt1) sıfırlanır, artırılır ve azaltılır. clkt1 saat sinyali dahili ya da harici saat kaynağından üretilebilir. Bunlar saat seçme bitlerinden denetlenir ve bunlar TCCR1B yazmacında bulunur. Eğer hiçbir saat kaynağı seçilmezse zamanlayıcı durur ve saymayı bırakır. Her durumda da TCNT1 değeri işlemci tarafından okunup yazılabilir. İşlemci yazma işlemi bütün zamanlayıcı sayma ve sıfırlama işlemlerini geçersiz kılar. Yani öncelik sırası işlemcidedir.

Sayma sırası dalga formu üreteci modu bitlerinin değiştirilmesiyle belirlenir. Bunlar TCCR1B ve TCCR1A yazmaçlarında bulunur. Zamanlayıcının sayımı ile dalga formlarının nasıl üretildiği arasında sıkı bir bağlantı vardır. Zamanlayıcı taşma bayrağı TIFR1 yazmacı içerisinde bulunur. Bu bit işlemci kesmesi oluşturmak için kullanılır.

Sıradaki derste diğer zamanlayıcı ünitelerini anlatmakla dersimize devam edeceğiz.

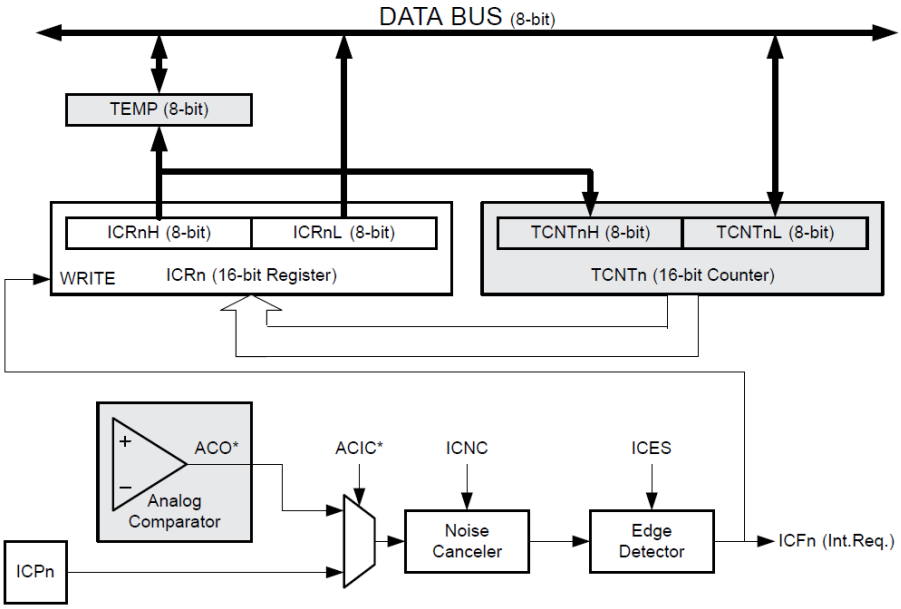
TC1 Giriş Yakalama Ünitesi (Input Capture Unit)

Önceki yazımızda TC1 zamanlayıcısının sayıcı ünitesinden bahsetmiş ve dersi bitirmiştik. Şimdi geri kalan birimlerinden bahselim ve sonrasında yazmaçlara geçelim.

Giriş Yakalama Ünitesi (Input Capture Unit)

TC1 zamanlayıcısı harici olayları birleştiren ve bunlara geçen süre zarfında zaman damgası veren bir ünitedir. Harici sinyal tek ya da çoklu olayları belirtir ve ICP1 ayağından uygulanır. Alternatif olarak analog karşılaştırıcı ünitesinden de yararlanılabilir. Bu zaman damgası frekans, görev döngüsü (duty-cycle) ve diğer uygulamaları hesaplamada kullanılabilir. Alternatif olarak zaman damgaları olay kütüğü (log) oluşturmada kullanılabilir.

Giriş yakalama ünitesi aşağıdaki blok diyagramında gösterilmiştir. Gri boyalı parçalar giriş yakalama ünitesinin asıl elemanlarından değildir. Yazmaç ve bitlerdeki “n” harfi zamanlayıcının numarasını temsil eder.



Giriş yakalama ayağında (ICP1) bir mantıksal seviye değişimi olduğunda (olay gerçekleştiğinde) ya da analog karşılaştırıcı çıkışında bir değişiklik olduğunda (ACO) bu değişiklik kenar algılayıcısı tarafından saptanır ve yakalama tetiklenir. 16 bitlik sayaç değeri (TCNT1) giriş yakalama yazmacına (ICR1) yazılır. Giriş yakalama bayrak biti (ICF) TCNT1 değerinin kopyalandığı aynı saat çeviriminde bir (1) konumuna geçer. Eğer TIMSK1 yazmacındaki ICIE biti bir (1) yapılırsa giriş yakalama bayrak biti giriş yakalama kesmesini yürütür. ICF1 bayrağı kesme yürüdükten sonra otomatik olarak sıfırlanır. ICF bayrağı alternatif olarak yazılımla bir (1) yazılarak da sıfırlanabilir.

Giriş yakalama yazmacının (ICR1) değerini okumak için öncelikle bunun düşük baytını (ICR1L) okumak gerekir. Sonrasında ise yüksek bayt okunur (ICR1H). Düşük bayt okunduğu zaman yüksek baytın değeri TEMP yazmacına aktarılır.

ICR1 yazmacı ancak dalga formu üreteci modunda yazılabilir. Bu mod ICR1 yazmacının değerini sayacın TOP değeri olarak belirler. Öncelikle dalga formu üreteci yazmaçlarına değer yazılmalıdır sonra ise ICR1 yazmacına istenilen değer yazılmalıdır. ICR1 yazmacına veri yazarken öncelikle yüksek bayt ICR1H yazmacına sonra ise düşük bayt ICR1L yazmacına yazılmalıdır. Önceki yazıda 16-bitlik yazmaçlara erişimden bahsetmiştik.

Giriş Yakalama Tetikleme Kaynağı

Giriş yakalama ünitesinin ana tetikleme kaynağı ICP1 ayağıdır. Ayrıca analog karşılaştırıcının çıkışı da tetikleme kaynağı olarak kullanılabilir. Analog karşılaştırıcı analog karşılaştırıcı denetleme ve durum yazmacı (ACSR) yazmacının ACIC biti ile tetikleme kaynağı olarak seçilebilir. Tetikleme kaynağını değiştirmek bir yakalamayı tetikleyebilir. O yüzden giriş yakalama bayrak biti değişimden sonra sıfırlanmalıdır.

Giriş yakalama ayağı (ICP1) ve Analog karşılaştırıcı çıkışı (ACO) girişleri T1 ayağında olduğu gibi aynı teknikle örneklendirilir. Kenar algılayıcısı da özdeştir. Gürültü önleyici etkinleştirildiğinde kenar yakalayıcısından önce ek bir mantık işlemi yürütülür. Böylelikle sistemi dört sistem saat çevirimi daha bekletir. Gürültü engelleyicinin girişi ve kenar algılayıcı zamanlayıcı dalga formu üreteci moduna alınmadığı sürece etkindir. Giriş yakalama ICP1 ayağını kontrol ederek yazılım ile de yapılabilir.

Gürültü Engelleyici

Gürültü engelleyici basit dijital filtreleme işleminde gürültünün engellenmesi için kullanılır. Gürültü engelleyici girişi dört adet örnek alır ve bu dört örneğin kenar algılayıcı tarafından kullanılan çıkışı değiştirecek kadar eşit olmasını denetler. Gürültü engelleyici Giriş Yakalama Gürültü Önleyici bitini değiştirmekle etkin hale getirilebilir. Bu ICNC biti TCCR1B yazmacında bulunmaktadır. Etkin hale getirildiğinde gürültü engelleyicinin çalışması için ekstradan dört saat çevirimi daha kullanıldığından bu kadar bir bekleme söz konusudur.

Giriş Yakalama Ünitesini Kullanmak

Giriş yakalama ünitesini kullanırken en önemli mesele gelen sinyali işleyecek yeterli işlemci kapasitesidir. İki sinyal olayı arasındaki zaman çok önemlidir. Eğer işlemci ICR1 yazmacındaki değeri bir sonraki yakalama olayı gerçekleştiremezse ICR1 üzerine yeni değer yazılır ve yakalama sonucu hatalı olur. Giriş yakalama kesmesini kullanırken ICR1 yazmacı kesme rutininde mümkün olduğu kadar erken okunmalıdır. TOP değerinin etkin olarak değiştirilmesi giriş yakalamada tavsiye edilmez. Harici sinyalin görev döngüsünü ölçmek için tetikleyici kenar her yakalamada değişmelidir. Kenar algılamayı ICR1 yazmacı değiştirildikten hemen sonra yapmak gerekir. Kenar değişiminden sonra Giriş Yakalama Bayrak Biti (ICF) yazılım tarafından sıfırlanmalıdır. (yani mantıksal bir (1) yazılması gerekir.) Frekans ölçümünde ICF bayrağını sıfırlamak gerekmez.

TC1 Üniteleri

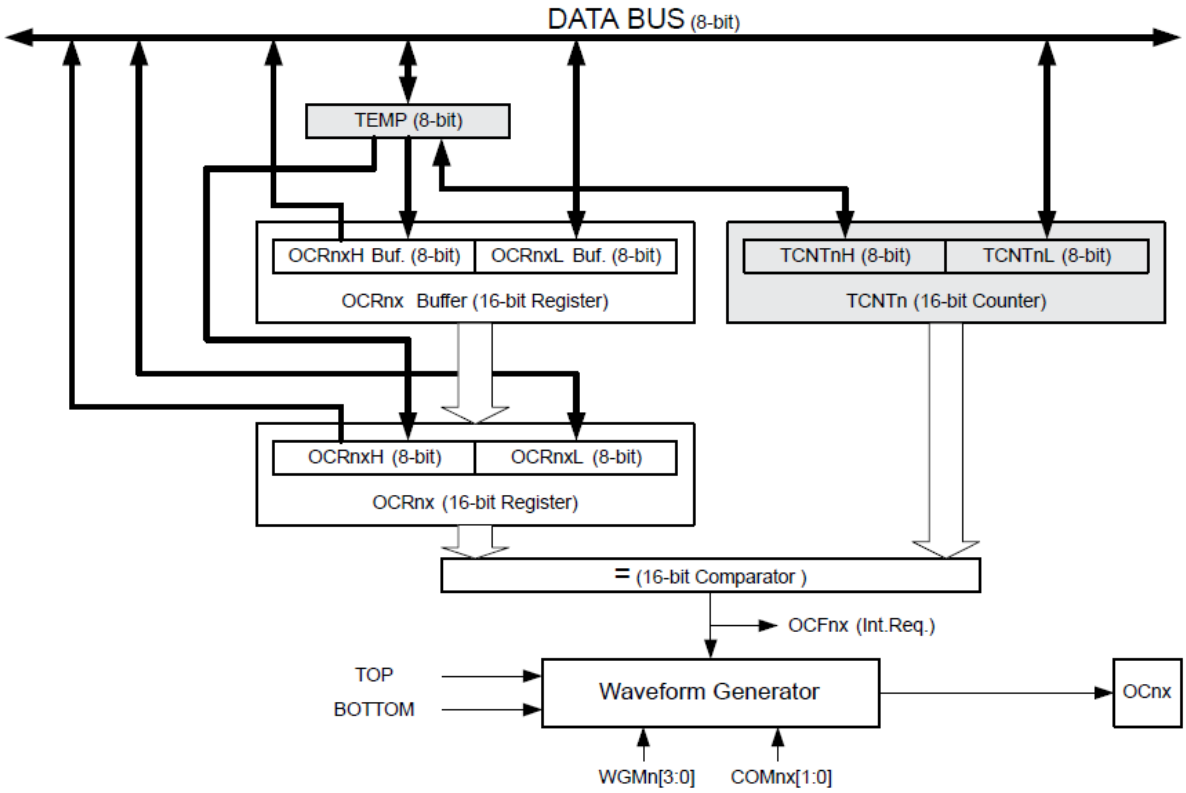
TC1 zamanlayıcısı başlığı altında zamanlayıcı ünitelerini teknik veri kitapçığından anlatmaya devam ediyoruz.

Çıkış Karşılaştırma Üniteleri

16-bit karşılaştırıcı devamlı olarak TCNT1 ve çıkış karşılaştırma yazmacını (OCR1x) karşılaştırır. Eğer TCNT, OCR1x'e eşit ise karşılaştırıcı eşleşme sinyali verir. Bu eşleşme

sinyali çıkış karşılaştırma bayrak bitinde yani TIFR1 yazmacının OCFx bitinde görülür. OCFx bayrak biti kesme yürüdüğünde otomatik olarak sıfırlanır. Buna yazılımla bir (1) yazarak sıfırlamak da mümkündür. Dalga formu üreteci bu eşleşme sinyalini çıkış üretmek için kullanır. TOP ve BOTTOM sinyalleri dalga boyu üreteci tarafından uç durumları denetlemek için bazı çalışma modlarında kullanılır.

Figure 20-4. Output Compare Unit, Block Diagram



Resim: ATmega328P – Microchip Technology , sf. 158, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 25.08.2018

Yukarıda blok diyagramı verilen çıkış karşılaştırma ünitesinde griye boyalı kısımlar ünitenin gerçek elemanları değildir.

OCR1x yazmacı toplamda 20 PWM modundan birini kullandığı zaman çift tamponlu modda olur. Normal ve karşılaştırmadan sonra temizle (CTC) modunda çift tamponlama devre dışıdır. Çift tamponlama etkin olduğunda işlemcinin OCR1x tampon yazmacına erişimi vardır. Çift tamponlama devre dışı olduğunda işlemci bu yazmaca doğrudan erişir.

OCR1x (Tampon veya karşılaştırma) yazmacındaki değer ancak bir yazma işlemi ile değiştirilebilir. Zamanlayıcı TCNT1 ve ICR1 yazmacındaki olduğu gibi otomatik olarak güncellemez. OCR1x yazmacının yüksek bayt verisi TEMP yazmacından okunmaz. Burada yüksek bayt önce yazılmalıdır. (OCR1xH)

PWM modunda değilken FOC1x bitine bir (1) yazarak karşılaştırıcının çıkışı zorla yürütülebilir. Zorla yürütünce OCF1x bayrağı bir (1) konumuna gelmez ve zamanlayıcıyı sıfırlamaz.

TCNT1 yazmacına işlemci tarafından gerçekleştirilen tüm yazım işlemi sonraki saat çevrimindeki karşılaştırma eşleşmesini engeller.

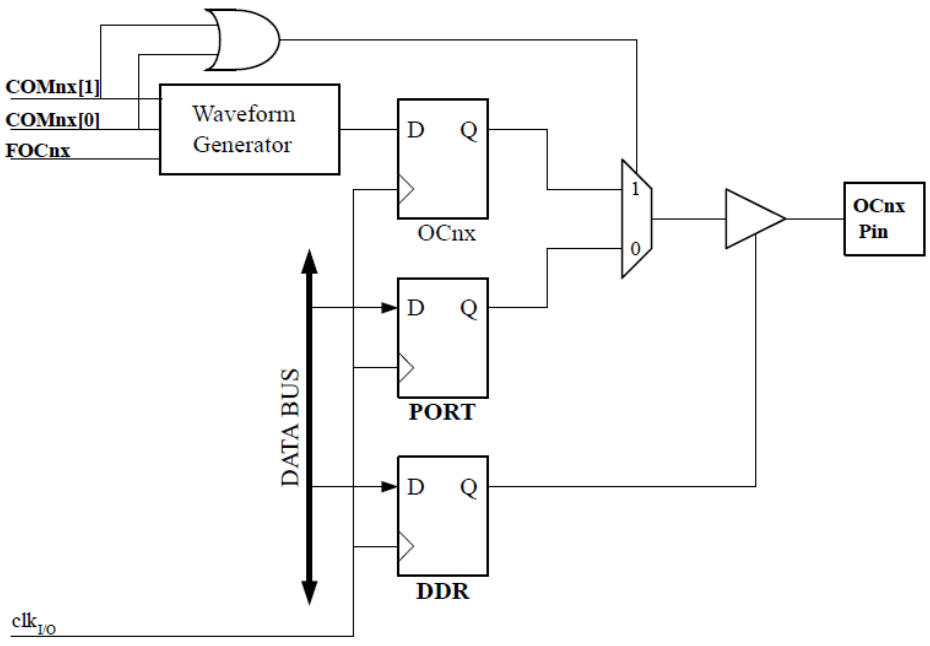
Çıkış karşılaştırma ünitesini kullanmak

TCNT1'e yazılan bir veri bir saat çevirimi kadar karşılaştırma eşleşmelerini durdurur. Eğer TCNT1 yazmacına yazılan veri OCR1x değerine eşit ise karşılaştırma eşleşmesi kaçırılır ve yanlış dalga formu üretimine sebep olur. TCNT1 yazmacına PWM modunda TOP değer yazılmamalıdır. TOP için karşılaştırma eşleşmesi görmezden gelinir ve sayaç 0xFFFF değerine doğru devam eder. Aynı şekilde sayaç aşağı doğru sayıyorsa BOTTOM değeri sayaca yazılmamalıdır.

OC1x ayarı DDR yazmacında ayak çıkış olarak tanımlanmadan önce yapılmalıdır. OC1x değerini ayarlamanın en kolay yolu FOC1x bitlerini normal modda kullanmaktır. OC1x yazmacı dalga formu değişikliğinde bile değeri korur.

Karşılaştırma Eşleşme Çıkış Ünitesi

Çıkış karşılaştırma modu biti (TCCR1A yazmacındaki COM1x[1:0] bitleri) iki işleve sahiptir. Dalga formu üreteci bu bitleri sonraki çıkış karşılaştırma eşleşmesinde (OC1x) belirleyici bit olarak kullanır. İkinci olarak bu bitler OC1x ayağının çıkış kaynağını belirlemeye yarar.Aşağıdaki şemada TCCR1A yazmacının COM1x [1:0] bitlerinin ayarlarına göre değişen mantıksal devre gösterilmiştir.



Resim : ATmega328P – Microchip Technology , sf. 160, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 25.08.2018

Burada “n” ile belirtilen aygıt numarasıdır. “x” ile belirtilen ise çıkış karşılaştırma ünitesidir (A/B). Eğer TCCR1A.COM1x bitleri bir yapılırsa genel giriş ve çıkış işlevi çıkış karşılaştırma (OC1x) tarafından geçersiz kılınır. OC1x ayağının giriş ve çıkış denetimi yine DDR yazmacındadır. OC1x kullanılmadan önce DDR yazmacının uygun düşen ayağı çıkış yapılmalıdır.

Karşılaştırma çıkış modu ve dalga formu üretimi

Dalga boyu üreteci TCCR1A yazmacındaki COM1x bitlerini normal, CTC ve PWM modlarında farklı olarak kullanır. Tüm modlar için bu bitleri sıfır (0) yapmak herhangi bir işlev olmayacağını belirtir. Bu bitleri değiştirmek bir sonraki karşılaştırma eşleşmesine kadar faaliyete geçirmeyecektir. PWM olmayan modda TCRR1C yazmacının FOC1x biti ile bu durum hemen gerçekleştirilebilir.

Buraya kadar TC1 zamanlayıcısının iç yapısını ve ünitelerini anlattık. Bir seferde okuyup anlamanızı beklemiyoruz. Önemli olan okuyup zamanlayıcılar hakkında fikir edinmenizdir. Sonraki derste zamanlayıcı çalışma modlarından bahsedeceğiz ve yine datasheet üzerinden anlatmaya devam edeceğiz.

Zamanlayıcı Çalışma Modları

Önceki yazılarımızda zamanlayıcıların yazmaçlarını ve iç yapısını anlatmıştık. Bu yazıda çalışma modlarından bahsedeceğiz ve sonrasında uygulamalara geçeceğiz. TC1 zamanlayıcısının yazmaçlarını da anlatmayı düşünsek de TC0 zamanlayıcısının yazmaçları ile hemen hemen aynı olduğu ve aradaki farklara ise TC1 zamanlayıcısının iç yapısını anlatırken değindiğimiz için bu konuyu atlayacağız. Eğer isteyen olursa Atmega 328p teknik veri kitapçığına (datasheet) başvurabilir. Yazmaç adları ve açıklamaları TC0 yazmacı ile benzer olduğu için TC0 yazmaçları hakkında anlattığımız çoğu şey TC1 yazmaçları için de geçerlidir. Aynı şekilde TC0 ve TC1 hakkında anlattığımızın çoğu TC2 için de geçerlidir. Şimdi çalışma modlarından bahsedelim ve konumuzu bitirelim.

Çalışma modları zamanlayıcı ve sayıcı ile çıkış karşılaştırma ayaklarının nasıl çalışacağını belirler. Bu modları seçmek için Dalga formu üretici modu bitleri (WGM1[3:0]) ile karşılaştırma çıkış modu bitleri (TCCR1A.COM1x[1:0]) kullanılır. Bu bitlerin durumuna göre zamanlayıcımız farklı modlarda çalışır.

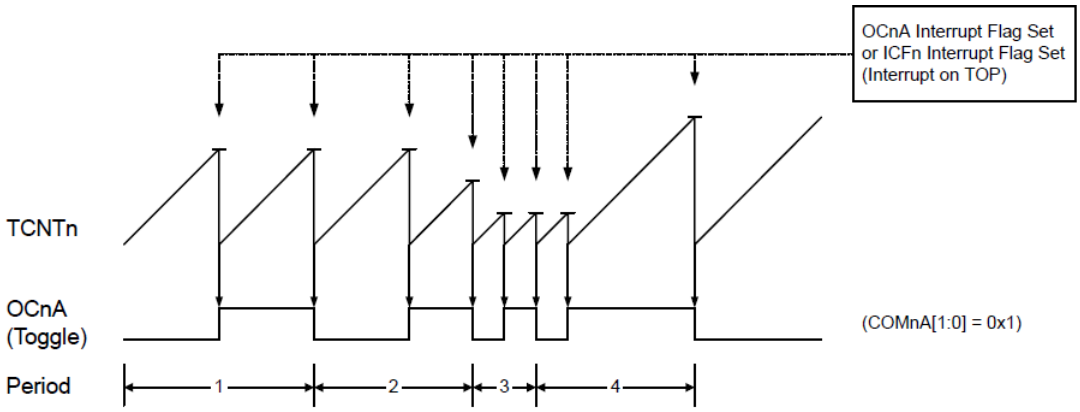
Normal Mod

Bu mod en basit çalışma modu olup WGM1[3:0] bitlerinin tamamı sıfır olunca etkin hale gelir. Bu modda sayma hep yukarıya doğrudur ve sayıcı sıfırlama yürütülmez. Sayıcı azami 16-bit değere ulaşınca otomatik sıfırlanır ve 0’dan itibaren tekrar saymaya başlar. TCNT1 yani sayaç verisi yazmacı sıfırlandığında zamanlayıcı taşma bayrak biti bir konumuna

geçer. Zamanlayıcı taşma kesmesinde bu bayrak biti sıfırlanır. Zamanlayıcı çözünürlüğü yazılımsal olarak artırılabilir. Yeni sayaç değeri istenilen bir zamanda yazılabilir.

Karşılaştırma Eşleşmesinde Sıfırlama Modu (CTC)

Bu moda WGM1[3:0] bitlerinin 0x4 yapılmasıyla geçilir. OCR1A ve ICR1 yazmaçları sayaç çözünürlüğünün manipulasyonunda kullanılır. Sayaç değeri OCR1A ya da ICR1 yazmacına ulaştığında sıfırlanır. OCR1A ve ICR1 yazmaçları sayacın üst değerini belirler bundan dolayı da çözünürlüğünü etkiler. Bu mod karşılaştırma eşleşmesi çıkışı frekansını denetimini sağlar. Ayrıca harici olayları sayma işini de kolaylaştırır. CTC modunun zaman diyagramı aşağıda verilmiştir. TCNT1 değeri (sayaç değeri) OCR1A ya da ICR1 ile eşleşene kadar artmaya devam eder. Sonrasında TCNT1 değeri sıfırlanır.



Eğer sayıcı değeri TOP değere ulaşırsa kesme yürütülebilir. Bunlar OCF1A'nın ve ICF1'in bayrak bitlerini kullanılarak yapılır. Eğer kesmeler etkin ise kesme fonksiyonu TOP değeri güncellemek için kullanılabilir.

CTC modunda dalga formu çıkışı isteniyorsa OC1A çıkışının mantıksal değeri her eşleşmede değiştirilmelidir. OC1A değeri o ayak çıkış yapılmadığı sürece okunamaz. O yüzden DDR_OC1A = 1 olmalıdır. Üretilen dalga formunun azami frekansı foc1a = fclk_io/2 olabilir. Dalga formu frekansı aşağıdaki formülde verilmiştir.

$$f_{\mathrm{OCnA}} = \frac{f_{\mathrm{clk\_I/O}}}{2 \cdot N \cdot (1 + \mathrm{OCRnA})}$$

Burada f clk_i/o işlemci frekansı, N ön derecelendirici faktörü, n ise aygıt numarasıdır. Normal modda olduğu gibi zamanlayıcı sayıcı TOV bayrak biti sıfırlanma sırasında bir (1) konumuna geçer.

Hızlı PWM Modu

Hızlı PWM modu yüksek frekansta PWM dalgası oluşturmayı sağlar. Sayaç BOTTOM konumundan TOP konumuna kadar saymaya devam eder ve sonrasında tekrar BOTTOM konumundan saymaya başlar. Terslemesiz karşılaştırma çıkış modunda çıkış karşılaştırma yazmacı (OC1x) TCNT1 ve OCR1x yazmaçları arasında eşleşme olduğunda BOTTOM değerine sıfırlanır. Terslemeli karşılaştırma modunda ise çıkış karşılaştırma eşleşmesine eşlenir ve BOTTOM değerine sıfırlanır. Yüksek frekanslı PWM modu güç regülasyonu, doğrultma ve DAC uygulamaları için uygundur. Yüksek frekans ayrıca daha küçük boyutlu dış parça kullanımına olanak tanıdığı için maliyetleri düşürür.

Hızlı PWM'nin çözünürlüğü 8, 9 ya da 10-bit 'e ayarlanabilir. Ya da isteniyorsa kesin çözünürlük ICR1 ve OCR1A yazmaçlarıyla ayarlanabilir. Asgari çözünürlük ise 2 bittir. Azami çözünürlük ise 16 bittir. Bu durumda ICR1 ve OCR1A yazmaçları azami değere alınmalıdır. PWM çözünürlüğü aşağıdaki formülden hesaplanabilir.

$$R_{\mathrm{FPWM}} = \frac{\log(\mathrm{TOP}+1)}{\log(2)}$$

Faz Düzeltmeli PWM Modu

Faz düzeltmeli PWM modu yüksek çözünürlüklü faz düzeltmeli dalga formu üretimi olanağı sağlar. Bu sefer sayıcı BOTTOM (0) değerinden TOP değerine saymaya başlar ve ardından TOP değerinden BOTTOM değerine saymaya devam eder.

Faz ve Frekans Düzeltmeli PWM modu

Faz ve frekans düzeltmeli PWM modu yüksek çözünürlüklü faz ve frekans düzeltmeli PWM dalga formu olanacağı sağlar. Bu modları PWM konusunda daha ayrıntılı anlatacağımız için kısa kesiyoruz.

Buraya kadar zamanlayıcıların yazmaçlarını, iç yapısını ve çalışma modlarını anlattık. Sonraki derslerde bu konuyu toparlayacağız ve buraya kadar bahsettiğimiz bilgiler üzerinden konuyu anlatmaya devam edeceğiz.

TC1 Zamanlayıcı Örnek Kodu

Bu dersimizde teknik veri kitapçığı üzerinden anlattığımız zamanlayıcıları bu sefer örnek kodlar üzerinden anlatacağız. Kod üzerinden anlatılması anlaşılması için çok daha faydalı olacaktır. Yine önceden olduğu gibi kodu satır satır açıklayacağız. Şimdi ilk programımıza geçelim.

```
// Bu program 16Mhz saat hızında 1 saniyelik zamanlama işlemi yapar

#include <avr/io.h>
#include <avr/interrupt.h>


int main(void)
{
    OCR1A = 0x3D08;

    TCCR1B |= (1 << WGM12);
    // CTC Mod 4

    TIMSK1 |= (1 << OCIE1A);
    //Karşılaştırma eşleşmesi kesmesini etkinleştir

    TCCR1B |= (1 << CS12) | (1 << CS10);
    // Ön derecelendiriciyi 1024 olarak belirle ve zamanlayıcıyı aç


    sei();
    // kesmeler etkin


    while (1)
    {

    }
}

ISR (TIMER1_COMPA_vect)
{
    // Her saniyede bir yapılacak işlemler
}
```

Bu program 16MHz çalışma frekansında her saniyede bir zamanlayıcı kesmesini çalıştırır. Öncelikle kesmeleri kullanabilmek için **#include <avr/interrupt.h>** komutu ile başlık dosyası eklenmiştir.

OCR1A = 0x3D08; OCR1A yazmacını önceki derslerde karşılaştırıcı yazmacı olduğunu anlatmıştık. Bu yüzden bu yazmaca bir karşılaştırma değeri atamak lazımdır. Bu ise 16MHz hızda ve 1024 ön derecelendirme ile toplamda 1 saniyelik saat çevirim değeri olmalıdır. TC1 zamanlayıcısı 16-bit olduğu için 16 bitlik bir değer atanmıştır. OCR değeri aşağıdaki formüle göre hesaplanır

OCRn = [(saat hızı / ön derecelendirici değeri) * Saniye olarak istenilen zaman] – 1

TCCR1B |= (1 << WGM12); TCCR1B yazmacı zamanlayıcı ile ilgili ayar verilerini bulunduran yazmaçtı. Bu yazmacın WGM12 biti bir (1) yapıldığında diğer ayar bitleri sıfır olacağından CTC modunda çalışacaktır. Yani zamanlayıcı karşılaştırma değerine göre sayma işlemini yapacaktır.

TIMSK1 |= (1 << OCIE1A); Zamanlayıcı eşleşme kesmesi etkinleştirilir.

TCCR1B |= (1 << CS12) | (1 << CS10); Bu CS bitlerinin tamamı sıfır olduğunda zamanlayıcı çalışmıyordu. Bu bitleri değiştirmek hem ön derecelendirici modunu seçmemizi sağlayacaktır hem de zamanlayıcıyı çalıştıracaktır. Bu bitler ile beraber zamanlayıcı 1024lük ön derecelendirici modunda çalışmaya başlar.

sei(); Bu komut ile kesmeler etkinleştirilir. Kesmelerin sürekli etkin halde olması hassas işlemlerde doğru olmayabilir. Bölünmesini istemediğimiz kısımlarda kesmeleri kapatıp gerektiği yerde açmak uygun olacaktır.

ISR (TIMER1_COMPA_vect) Kesme gerçekleştiğinde bu fonksiyon içerisindeki komutlar çalıştırılır.

Bu programda zamanlayıcımız sıfırdan 0x3D08 değerine kadar her 1024 saat çeviriminde (Çünkü ön derecelendirici ile 1024'e böldük.) birer birer saymaya başlar. O değere ulaştığı zaman CTC modunda olduğu için otomatik sıfırlanır ve tekrar sıfırdan bu değere saymaya başlar. Bu sayma işlemini kesinlikle mikroişlemci yapmamaktadır. Mikroişlemci o sırada başka komutları işlerken zamanlayıcı ünitesi saymaya devam etmektedir. Zamanlayıcı ünitesi ne zaman belirtilen değere kadar saymayı bitirirse o zaman kesme yürütülerek mikroişlemcinin haberdar olması sağlanır.

Sonraki derslerde örnek kodları açıklayarak devam edeceğiz.

Frekans Sayıcı Programı İncelemesi

İnternette Arduino için yazılmış ama çoğunlukla AVR kodlarından oluşan benim de kullandığım bir frekans sayıcı programı vardı. Bu programı beğendiğim için ne kadar Arduino uyumlu olsa da AVR üzerinden anlatarak derslere ilave edeceğim. Bu program Arduino UNO'yu yani Atmega328P mikrodenetleyicisinin zamanlayıcı ünitelerini kullanarak D5 ayağından yani PD5 / T1 ayağından giriş alır. Belli bir zaman periyodunu kullanarak o ayağa gelen sinyalleri ölçerek frekansı bildirir. Uzun zaman periyotu daha fazla hassasiyet verecektir. Şimdi programı verelim ve sonrasında kodları tek tek açıklayalım.

```
// Frekans Sayıcı Örneği
// Yazar: Nick Gammon
// Tarih: 17th January 2012

// Giriş D5 ayağı

// Bu değerler ana program tarafından denetlenecektir.
volatile unsigned long timerCounts;
volatile boolean counterReady;

// sayaç rutini değerleri
unsigned long overflowCount;
unsigned int timerTicks;
unsigned int timerPeriod;

void startCounting (unsigned int ms)
```

```
  {
  counterReady = false;          // Zamanı daha gelmedi
  timerPeriod = ms;              // kaç milisaniye sayılacağı
  timerTicks = 0;                // kesme sayacını sıfırla
  overflowCount = 0;             // taşma henüz yok

  // TC1 ile TC2 yi sıfırla
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR2A = 0;
  TCCR2B = 0;

  // D5 ayağındaki olayları sayar.
  TIMSK1 = bit (TOIE1);   // TC1 taşmasında kesmeye götür

  // Timer 2 -1 ms ölçme aralığını verir.
  // 16 MHz saat (62.5 ns çevirim başı) - 128 ön derecelendirici
  //  Sayaç her 8 mikrosaniyede bir artar
  // Bundan 125 adet sayıldığında bize tam 1ms verir.
  TCCR2A = bit (WGM21) ;   // CTC modu
  OCR2A  = 124;            // 125'e kadar sıfırdan itibaren say

  // Timer 2 - her eşleşmede kesme (her 1 milisaniyede bir )
  TIMSK2 = bit (OCIE2A);   // TC2 kesmesini etkinleştir

  TCNT1 = 0;      // Her iki sayacı da sıfırla
  TCNT2 = 0;

  // Ön derecelendiricileri sıfırla
  GTCCR = bit (PSRASY);        // Ön derecelendiriciyi sıfırla
  // TC2'yi başlat.
  TCCR2B =  bit (CS20) | bit (CS22) ;  // Ön derecelendiriciyi 128'e ayarla
  // TC1 'i başlar.
  // T1 ayağından harici saat kaynağı. (D5). Yükselen kenarda sayım.
  TCCR1B =  bit (CS10) | bit (CS11) | bit (CS12);
  }  // startCounting fonksiyonu bitişi

ISR (TIMER1_OVF_vect)
  {
  ++overflowCount;               // Counter1 taşma sayısını say.
  }  //  TIMER1_OVF_vect bitimi


//*****************************************************************
//  Timer 2 kesme servisi donanım tarafından  her 1 ms de bir işletilir = 1000
Hz
//  16Mhz / 128 / 125 = 1000 Hz

ISR (TIMER2_COMPA_vect)
  {
  // sayaç değerini değişmeden hemen al
  unsigned int timer1CounterValue;
  timer1CounterValue = TCNT1;  // 16 bit yazmaç okunur.
  unsigned long overflowCopy = overflowCount;

  // zamanlama periyoduna ulaşıp ulaşmadığı kontrol edilir.
  if (++timerTicks < timerPeriod)
    return;  // henüz değil.

  // Eğer taşma kaçırılırsa
```

```
  if ((TIFR1 & bit (TOV1)) && timer1CounterValue < 256)
    overflowCopy++;

  // geçit zamanı bitişi, ölçmeye hazır.

  TCCR1A = 0;     // TC1 i durdur
  TCCR1B = 0;

  TCCR2A = 0;     // TC2 yi durdur
  TCCR2B = 0;

  TIMSK1 = 0;     // TC1 kesmesini devre dışı bırak
  TIMSK2 = 0;     // TC2 kesmesini devre dışı bırak

  // Toplam değeri hesapla
  timerCounts = (overflowCopy << 16) + timer1CounterValue;  // 65536 her taşma.
  counterReady = true;               // her saymadan sonra genel bayrağı
etkinleştir
  }  // TIMER2_COMPA_vect bitişi

void setup ()
  {
  Serial.begin(115200);
  Serial.println("Frequency Counter");
  } // setup bitişi

void loop ()
  {
  // TC0 kesmelerini durdur
  byte oldTCCR0A = TCCR0A;
  byte oldTCCR0B = TCCR0B;
  TCCR0A = 0;     // TC0'ı durdur
  TCCR0B = 0;

  startCounting (500);  // ölçüm için kaç milisaniye ayarlayacağını belirler
  while (!counterReady)
     { }  // Sayım bitene kadar devam et

  // sayılan değeri sayım aralığına göre artır ve frekansı ver
  float frq = (timerCounts *  1000.0) / timerPeriod;

  Serial.print ("Frequency: ");
  Serial.print ((unsigned long) frq);
  Serial.println (" Hz.");

  // TC0'ı yeniden başlat.
  TCCR0A = oldTCCR0A;
  TCCR0B = oldTCCR0B;

  // bitir
  delay(200);
  }   // döngü bitimi
```

Öncelikle loop() fonksiyonundan kodları incelemeye başlayalım.

```
 byte oldTCCR0A = TCCR0A;
 byte oldTCCR0B = TCCR0B;
```

Burada TCCR0 yazmaçlarında olan ilk değer oldTCCR0A ve oldTCCR0B değişkenlerine aktarılarak saklanır.

```
  TCCR0A = 0;
  TCCR0B = 0;
```

Burada TCCR0A ve TCCR0B yazmaçları sıfırlanarak zamanlayıcı durdurulur.

```
startCounting (500);
```

Frekans ölçme fonksiyonu 500 argümanı ile birlikte çalıştırılır. Bunun anlamı frekans ölçme fonksiyonu 500 milisaniye boyunca gelen frekansı ölçecek demektir. Şimdi startCounting() fonksiyonuna geçelim ve ondan sonra tekrar ana program döngüsüne dönelim.

```
void startCounting (unsigned int ms)
  {
  counterReady = false;         // Zamanı daha gelmedi
  timerPeriod = ms;             // kaç milisaniye sayılacağı
  timerTicks = 0;               // kesme sayacını sıfırla
  overflowCount = 0;            // taşma henüz yok
```

Burada startCounting fonksiyonunun aldığı argüman “ms” adıyla adlandırılır. “ms” değişkeninin kullanıldığı yer bizim milisaniye değerimiz olan 500 olacak. Önceden dört adet genel değişken program başında tanımlanmıştı. Şimdi bunlara fonksiyonun başlangıcında ilk değerlerin verildiğini görüyoruz. CounterReady yani sayacın hazır olduğu false durumuna getirilmiştir. Böylelikle sayacın meşgul olduğu belirtilir. timerPeriod değişkenine ise bizim argüman olarak aldığımız ms değişkeninin değeri aktarılmıştır. 500 değeri şimdi de timerPeriod değişkenine geçmiştir. timerTicks değişkeni ise zamanlayıcının kaç defa attığını yani kesmeye geçtiğini sayma verisini tutan değişkendir. overflowCount ise kaç defa taşma gerçekleştiğinin sayım verisini tutan değişken olarak belirtilmiştir. Bu değişkenlerin eski değerleri yeni sayımda önemli olmadığı için hepsi sıfırlanmıştır.

```
  // TC1 ile TC2 yi sıfırla
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR2A = 0;
  TCCR2B = 0;
```

TC1 ve TC2 zamanlayıcılarının kontrol yazmaçları sıfırlanmıştır. Böylelikle zamanlayıcıların tüm özellikleri sıfırlanıp devre dışı bırakılmıştır.

```
// D5 ayağındaki olayları sayar.
  TIMSK1 = bit (TOIE1);   // TC1 taşmasında kesmeye götür
```

TIMSK1 yazmacının TOIE1 biti bir (1) yapılmıştır. Buradaki bit() fonksiyonu AVR’deki BV_() makrosu ile aynı işlevi görmektedir. TOIE0 bitine bir (1) yazıldığında Zamanlayıcı/Sayıcı Taşma Kesmesi etkin hale gelir.

```
// Timer 2 -1 ms ölçme aralığını verir.
  // 16 MHz saat (62.5 ns çevirim başı) - 128 ön derecelendirici
  //  Sayaç her 8 mikrosaniyede bir artar
  // Bundan 125 adet sayıldığında bize tam 1ms verir.
  TCCR2A = bit (WGM21) ;   // CTC modu
  OCR2A  = 124;            // 125'e kadar sıfırdan itibaren say
```

TCCR2A yani TC2 zamanlayıcısının A isimli kontrol yazmacındaki WGM21 biti bir (1) konumuna getirilir. Böylelikle zamanlayıcı CTC modunda çalışmış olur. Bunun için karşılaştırma değeri olan OCR2A yazmacına ise 124 değeri atanmıştır.

```
// Timer 2 - her eşleşmede kesme (her 1 milisaniyede bir )
  TIMSK2 = bit (OCIE2A);   // TC2 kesmesini etkinleştir
```

TC2 sayıcısının TIMSK2 yazmacındaki OCIE2A biti bir (1) yapılmıştır. OCIE0A bitine bir (1) yazıldığına Zamanlayıcı/Sayıcı karşılaştırma eşleşmesi kesmesi etkin hale gelir. Böylelikle yukarıda OCR2A yazmacına atanan 124 değeri ile eşleşme sağlanırsa kesme yürütülür.

```
TCNT1 = 0;      // Her iki sayacı da sıfırla
TCNT2 = 0;
```

Her iki zamanlayıcının sayaç değerlerini sayaç değeri yazmacına sıfır (0) atayarak sıfırlıyoruz.

```
 // Ön derecelendiricileri sıfırla
  GTCCR = bit (PSRASY);        // Ön derecelendiriciyi sıfırla
```

GTCCR yazmacının PSRASY biti bir yapıldığında ön derecelendirici sıfırlanır.

```
 // TC2'yi başlat.
  TCCR2B =  bit (CS20) | bit (CS22) ;  // Ön derecelendiriciyi 128'e ayarla
  // TC1 'i başlar.
  // T1 ayağından harici saat kaynağı. (D5). Yükselen kenarda sayım.
  TCCR1B =  bit (CS10) | bit (CS11) | bit (CS12);
```

Buraya kadar zamanlayıcıları ayarladık, sıfırladık ve bazı değişkenlere değer atadık. Zamanlayıcıları ise şimdi bu kodlarla başlatıyoruz ve ardından kesme fonksiyonları ile diğer işlemleri yapıyoruz. Öncelikle TCCR2B yazmacının CS20 ve CS22 bitleri bir (1) yapılır. Böylelikle TC2 zamanlayıcısının ön derecelendiricisi 128'e ayarlanmış olur. Şimdi ise 16 bitlik TC1 zamanlayıcısının giriş ayağından frekans ölçmemiz gerekli. Bunun için TCCR1B yazmacının CS10, CS11 ve CS12 bitleri bir (1) yapılır. Böylelikle zamanlayıcının sayması için gereken saat frekansı T1 ayağından (PD5) verilmiş olur. Verilen saat frekansına göre yükselen kenarda sayım yapılır. Asıl frekans ölçüm verisinin depolandığı yer T1 zamanlayıcısının sayaç yazmacıdır. Bizim fonksiyonumuz bundan ibaret olsa da TC1 zamanlayıcısının dolup boşalan sayacı bizim için bir anlam ifade etmez. O veriyi işleyip yorumlamak için diğer komutlara ihtiyacımız vardır. Bunu da kesmeler ile yapıyoruz. Şimdi kesme fonksiyonlarına göz atalım.

```
ISR (TIMER1_OVF_vect)
  {
  ++overflowCount;               // Counter1 taşma sayısını say.
  }  //  TIMER1_OVF_vect bitimi
```

TIMSK1 = bit (TOIE1); komutuyla TC1 zamanlayıcısını taşma sağlandığında kesmeye götürmesini söylemiştik. Şimdi ise TIMER1_OVF_vect bu kesmede ne yapılacağını belirlememiz için burada kullanılmıştır. Her taşma gerçekleştiğinde taşma sayıcı değişkenin değeri bir artırılacaktır. Bu taşma verisi kullanılmak üzere şimdilik saklanacaktır.

```
ISR (TIMER2_COMPA_vect)
  {
  // sayaç değerini değişmeden hemen al
  unsigned int timer1CounterValue;
  timer1CounterValue = TCNT1;  // 16 bit yazmaç okunur.
  unsigned long overflowCopy = overflowCount;
```

Frekans değerini dışarıdan alıp depolayan TC1 yazmacının yanında süre hesabını yapan TC2 yazmacı vardı. Bu yazmaca OCR2A = 124; komutu ile önceden karşılaştırma değerini atayıp bu karşılaştırma gerçekleştiğinde ise kesmeye götürmesini söylemiştik. Şimdi bu kesme fonksiyonunda neler yapılacağı ise burada belirtilmiştir. Öncelikle teknik veri sayfasında belirtildiği üzere kesmeden sonra ilk öncelikle sayaç değerini okumamız gerekiyordu. Bunun için unsigned int cinsinde timer1CounterValue değişkeni tanımlanmıştır. Ardından sayaç verisini içinde barındıran TCNT1 yazmacı bu değişkene aktarılmıştır. TCNT1 yazmacı önceden de bahsettiğimiz üzere TC1 zamanlayıcısının sayaç yazmacıydı. Bunun ardından overflowcopy adında bir değişken tanımlanmış ve taşma sayımını yapan değişken olan overflowCount'un değeri buna aktarılmıştır.

```
// zamanlama periyoduna ulaşıp ulaşmadığı kontrol edilir.
if (++timerTicks < timerPeriod)
return; // henüz değil.
```

TC2 karşılaştırma eşleşme kesmesi her bir (1) mili saniyede bir yürütüldüğü için bizim timerPeriod olarak tanımladığımız ve kaç mili saniyelik bir ölçüm gerçekleştirileceğini belirlediğimiz değere ulaşılmadıkça bu ölçüm yapılmayacaktır. Öncelikle timerTicks yazmacı her bir mili saniyede yürütülen kesmeden dolayı bir artırılır ve timerPeriod değişkeni ile karşılaştırılır. ++ operatörünün başta olması işlem önceliği için önemlidir. return; komutu ile kesme fonksiyonundan çıkılır ve diğer kodlar işletilmez.

```
// Eğer taşma kaçırılırsa
if ((TIFR1 & bit (TOV1)) && timer1CounterValue < 256)
overflowCopy++;
```

Kesme fonksiyonuna girildiği zaman mevcut taşma durumu kaçırılırsa TIFR1 yazmacındaki taşma bayrak biti olan TOV1 denetlenir overflowCopy değişkeni bir artırılır. Normalde TC1 yazmacının özel taşma kesmesi fonksiyonu ile bu taşma verisi artırılıyordu.

```
// geçit zamanı bitişi, ölçmeye hazır.

  TCCR1A = 0;     // TC1 i durdur
  TCCR1B = 0;

  TCCR2A = 0;     // TC2 yi durdur
  TCCR2B = 0;

  TIMSK1 = 0;     // TC1 kesmesini devre dışı bırak
  TIMSK2 = 0;     // TC2 kesmesini devre dışı bırak
```

Şimdi bütün sayaçlar ve kesmeleri ölçümün hesaplanması üzere devre dışı bırakılır.

```
 // Toplam değeri hesapla
  timerCounts = (overflowCopy << 16) + timer1CounterValue;  // 65536 her taşma.
  counterReady = true;              // her saymadan sonra genel bayrağı
etkinleştir
```

Şimdi toplam sayılan frekansı hesaplamak için kaç kere taşma yapıldıysa bu değerin bitleri 16 kere sola kaydırılır. Bu işlem mevcut taşma değerini 65536 değeri ile çarpmaya eşdeğerdir. Ardından timerCounts değişkenine bu değer aktarılır. Sonrasında ise sayacın işlemi bitirdiği ve hazır olduğu anlamında counterReady değeri "true" yapılır.

Frekans ölçme ile ilgili tüm fonksiyonlar ve kesme fonksiyonlarını anlatmayı bitirdik. Şimdi ana programa dönelim ve frekansın nasıl hesaplandığına bakalım.

```
float frq = (timerCounts * 1000.0) / timerPeriod;
```

Frekans değeri timerCounts değişkeninin 1000 ile çarpımının timerPeriod değişkeninin değerine bölünmesiyle elde edilir. 1000 ile çarpılmasının sebebi her 1 mili saniyede bir TC2 eşleşme kesmesinin yürütülmesidir.

```
  // TC0'ı yeniden başlat.
  TCCR0A = oldTCCR0A;
  TCCR0B = oldTCCR0B;
```

Bu kodlarla ise TCCR0A ve TCCR0B değerlerinin ilk halleri tekrar bu yazmaçlara yüklenir ve sayma işlemi yeniden başlar. Görmezden geldiğimiz Arduino kodları bizim işimize yaramasa da buraya kadar incelediğimiz kodlar Arduino kodu olmayıp AVR programlamada geçerli kodlardır. O yüzden AVR programcıları için faydalı olacağına inandığım bu programı satır satır inceledim.

Zamanlayıcı ve Sayıcıların Özeti

Atmega mikrodenetleyicilerdeki zamanlayıcı ünitelerinin oldukça karmaşık olduğunu anlatmamıza gerek yoktur. On derstir anlatmaya çalıştığımız ve daha bitiremediğimiz bu konu kolay kolay kavranacak bir konu da değildir. Basit bekleme işlemlerinden karmaşık PWM işlemlerine kadar kullanılan bu üniteler prensipte basit fakat uygulamada zordur.

Buraya kadar anlatmadığımız konulardan biri de TC2 zamanlayıcısının asenkron olarak kullanımı ve gerçek zaman saati (RTC) uygulamasıydı. Ayrıca PWM konusuna daha geçiş yapmayıp zamanlayıcıları normal ve CTC modda anlatmaya devam ettik. Bunları şimdilik ertelemek zorundayız.

Zamanlayıcılar ana programdan bağımsız olarak işleyebilen ve mikroişlemciden bağımsız çalışan aygıtlardır. Mikroişlemci ile bağlantıları kesmeler ve yazmaçlardır. Kesmeler mikroişlemci üzerinde oluşturdukları etki, yazmaçlar ise mikroişlemcinin denetlediği birimlerdir. Kesme durumunda mikroişlemciyi bir işi yapmaya zorlarken geri kalan işlemlerde mikroişlemci yazmaçlar vasıtası ile denetleme ve okuma/yazma işlemlerini yapar.

Zamanlayıcıların ana parçaları sayaç değeri yazmacı ve karşılaştırma değeri yazmaçlarıdır. Buradaki değerler üzerinden sayaçlar çeşitli işlemleri yapar. Ayrıca ayar yazmaçları ile zamanlayıcının nasıl çalışacağı belirlenir. Giriş ve çıkış ayakları ile zamanlayıcılar dış dünya ile iletişime geçer.

Zamanlayıcılar çalışmak için mikroişlemciler gibi belli bir saat frekansına ihtiyaç duyar. AVR zamanlayıcıları bunu işlemcinin saat hızından edinir. Böylelikle işlemci ile aynı hızda çalışan bir zamanlayıcı elde etmiş oluruz. Bunu ön derecelendirici (prescaler) adı veren yapı ile bölüp yavaşlatmamız mümkündür. İşlemci hızında çalışan bir zamanlayıcı çoğu uygulama için oldukça hızlıdır.

Zamanlayıcıda zaman hesabı yapmak için **Zaman = 1 / Frekans** formülünü kullanırız. Örneğin 100Hz frekans girişinde zamanımız 1/100'den 0.01 yani 10 mili saniye olarak bulunur. Frekans/Zaman dönüşümü oldukça önemlidir ve sıkça kullanılır.

Zamanlayıcıda belli bir süre için kaça kadar sayması gerektiğini ise şu formülle buluruz,

Sayaç Değeri = (1/Frekans) / (1 / Saat Hızı) – 1

Örneğin saniyenin yirmide biri kadar bir süreyi 1MHz frekansta çalışan işlemcide elde etmek için şöyle bir formül uygulanır.

Sayaç Değeri = (1/20) / (1/1000000) – 1 = 0.05 / 0.000001 -1 = 50000 – 1 = 49999 Yani sayacımız 49999'a kadar sayıp sonra tekrar sıfırlamalıdır. Bu karşılaştırma değerini ise karşılaştırma yazmacına bizim yazmamız gereklidir. Böylelikle sayaç değer yazmacı ile karşılaştırma değer yazmacı eşitlendiğinde taşma meydana gelir ve kesme yürütülür.

Ön derecelendiriciler ile bu hızlı sayan zamanlayıcıyı yavaşlatıp esneklik sağlatırız. Bunun için gerekli yazmaç bitleri ayarlanmadan önce aşağıdaki formülteki gibi bir hesaplama yapılır.

Zaman Çözünürlüğü = 1 / (Frekans / Ön derecelendirici)

Örneğin ön derecelendirici olmaksızın (yani 1 iken) 1MHz saat hızında çözünürlük 1 mikrosaniye iken, ön derecelendirici 8 olduğunda 8 mikro saniye, 64 olduğunda 64 mikrosaniye, 256 olduğunda 256 mikrosaniye, 1024 olduğunda ise 1024 mikrosaniyedir. Yani saatimiz bir çevirimde bir tık atarken artık 1024 çevirimde bir tık atıyor. Böylelikle saati yavaşlatmış oluyoruz. Saatin her atmasında sayaç yazmacı değeri bir artmaktadır.

Şimdi ön derecelendiriciler ile beraber sayaç değeri hesaplamayı gösterelim. Aşağıdaki formülde hedeflenen sayaç değerini ön derecelendiricilerle elde etme gösterilmiştir.

İstenilen Sayaç Değeri = (1 / Hedef Frekans) / (Ön derecelendirici / Frekans) -1

Ya da

Sayaç Değeri = (Giriş Frekansı / Ön derecelendirici * Hedef Frekans) – 1

Şimdi 1 saniyelik bekleme için ön derecelendirici durumuna göre 1MHz'de ne kadar bir değer gerektiğine bakalım.

Ön derecelendirici değeri	İstenilen Sayaç Değeri
1	999999
8	124999

64	15624
256	3905.25
1024	975.5625

Küsüratlı sayılar kullanılamadığı için sadece bilgi vermek için yazılmıştır. Bu durumda mümkün olan tek değer 16 bit sayıcı için 64 değeridir.

Zamanlama işlemleri bu formüllere göre yapılır ve ön derecelendirici, zamanlama modu ayarları yazmaç bitlerine müdahale ile yapılır. Buraya kadar normal modu, CTC modunu (karşılaştırmalı mod), kesmeli CTC modunu örneklerle açıkladık. Temel zamanlayıcı fonksiyonlarını buraya kadar anlatabildiğimizi umuyoruz. Artık zamanlayıcıları burada bırakalım ve yeni konumuza geçelim.

Kesmeler

Kesmeler gömülü sistemlerde kullanılan en faydalı ve en temel özelliklerden biridir. Kesmeyi kısaca tarif etmek gerekirse işlemciyi dışarıdan müdahale ile durdurup özel bir fonksiyonu yürüten yapılardır, diyebiliriz. Kesmeler donanımsal olduğu için ek bir işlem gücü gerektirmez. Bu seviyedeki mikroişlemcilerin en büyük kısıtlılıklarından biri aynı anda iki işlemi birden yapamamalarıdır. Aynı anda iki işlem birden yürütülmediği için bunu dışarıdan denetleyecek ve işlem sırasını gözetecek bir yapıya ihtiyaç vardır. Örneğin mikrodenetleyiciye bir düğme bağladığımızda bunun durumunu yazılım boyu denetlememiz gerekecektir. Bu denetleme sürecinde farklı kodlar işletilmeyeceği gibi farklı kodların işletilmesi için de denetleme işleminin belli bir zaman yapılmaması gerekir. Bazı durumlarda mikrodenetleyici bu denetleme işlemleri ile gereksiz yere meşgul olur ve yavaşlar. Aynı zamanda bu denetleme işlemleri çoğalınca programın kararlılığı oldukça zayıflar. Bunun için bu durumları denetleyecek ve mikroişlemciyi dışarıdan yönlendirecek aygıta ihtiyaç vardır.

Bu durum aynı biz bir iş yaparken telefonun çalması ve bu dış uyarana karşılık bizim işimizi bölüp telefonu cevaplamamız gibidir. Telefonu sessize alıp telefonun çalıp çalmadığını her dakika başı denetlemek yerine telefonun sesine göre karar vermek nasıl mantıklı bir seçim ise mikrodenetleyicilerde kesmeleri kullanmak da o derece mantıklı bir seçim olacaktır. Basit programlarda ve her zaman ihtiyaç duyulmayan yerlerde (mikroişlemcinin işlem hızına güvenimizden dolayı) kesme kullanmayabiliriz. Ama kesmeleri muhakkak bilmemiz gereklidir.

Kesmeler donanım ve yazılım kesmeleri olarak ikiye ayrılsa da yazılım kesmeleri genellikle işletim sistemlerinin özel görevleri için kullanılıp AVR mikrodenetleyicilerde yer almaz. O yüzden AVR mikrodenetleyicilerde sadece donanım kesmeleri kullanılır. Bu kesmelerin adları, adresleri ve kaynakları (kesmeye sebep olan olay) mikrodenetleyicinin teknik veri sayfasında yer alır. C dilinde kesmeleri kullanmak oldukça kolaydır çünkü AVR'nin interrupts.h adında başlık dosyası mevcuttur. Burada kesme fonksiyonlarını yazıp sonrasında işletilmesini istediğimiz kodu arasına koyabiliriz. Kesme fonksiyonları özel bir adla adlandırılır, Kesme Servis Rutini (Interrupt Service Routine) olarak adlandırılan bu

fonksiyon kısaca ISR olarak ifade edilir. Kesme biti bayrakları her kesme yürüdükten sonra sıfırlandığı için tekrar sıfırlanmaya ihtiyacı yoktur.

Kesmelerin nasıl kullanıldığını önceden zamanlayıcı örnek kodlarında göstermiştik fakat sadece kullanımından ibaretti. Diğer birimlerde yazmaç olduğu gibi kesmelerde de yazmaçlar mevcuttur. Bu yazmaçları da konumuz süresince anlatacağız.

Bir kesmenin yürütülmesi için öncelikle şu adımlar izlenmelidir.

- AVR'nin genel kesme biti etkin olmalıdır. SREG yazmacındaki I biti bu görevi üstlenmektedir. Bu bit mikrodenetleyici açıldığında sıfır (0) konumundadır. Bizim sonradan kesmeleri etkinleştirmemiz gereklidir.
- Kesme kaynağının etkinleştirme biti etkin olmalıdır. Önceden anlattığımız gibi bu kesmeleri etkinleştirme bitleri birim yazmaçlarının içerisinde yer alabilmektedir. Bu bitler özel bir kesmeyi etkinleştirmek için kullanılır. Genel kesmelerin etkin olması tüm kesmelerin etkin olacağı anlamına gelmemektedir. O yüzden hangi kesmeyi kullanmak istiyorsak o kesmeyi elle etkinleştirmemiz gereklidir.
- Kesme şartı sağlanmalıdır. Yani etkinleştirdiğimiz kesmenin bir iş yapması gereklidir. Boş yere bir kesmeyi çalıştırmanın anlamı yoktur. Onu da kesme fonksiyonuna yazdığımız kodlarla yapıyoruız.
  AVR mikrodenetleyiciler kesmedeyken kesmeye gidemez. Kesme yürütüldüğü sırada kesmeler devre dışı bırakılır. Bu yüzden kesme içinde kesme gerçekleşmesi mümkün değildir. O yüzden kesme fonksiyonlarını çok uzun tutmamak atlanılan kesmelerin oluşmasını önleyecektir. Çoğu kesme fonksiyonu kısa görevleri yapmak için yürütüldüğünden böyle bir durumun olması pek fazla sıkıntı doğurmayacaktır.

Genel kesmeleri etkinleştirmek ve devre dışı bırakmak için avr/interrupt.h başlık dosyasını aldıktan sonra iki fonksiyon kullanıyoruz. sei() (SEt Interrupt Bit) genel kesmeleri etkinleştirirken cli() (CLear Interrupt Bit) genel kesmeleri devre dışı bırakır. Bazı hassas uygulamaların kesmelerle bölünmesini istemeyebiliriz. Bu durumda kesmeleri devre dışı bırakıp mikrodenetleyicinin o süre zarfında ancak bizim istediğimiz kodları işlemesini sağlayabiliriz.

Kesmeler yürütüldüğü zaman ana program yürümeyeceği için kısa zaman zarfında oldukça fazla kesme yürütmek ana programın yavaşlamasına sebep olur. Bunu dikkate alarak gereğinden fazla kesme kullanmamak lazımdır.

Kesme fonksiyonları ile ana fonksiyon arasında kullanılan değişkenler volatile veya global (genel) olmalıdır. Kesme fonksiyonları ana programdan ayrı yürüdüğü için ana programın içindeki değişkenlere erişimi yoktur. Şimdi örnek bir kesme fonksiyonunu verelim ve konumuzu bitirelim.

```
ISR ( kesme_vektoru )
{
komutlar;
}
```

İşte programa kesmeleri eklemek bu denli kolaydır. Sonraki dersimizde teknik veri sayfası üzerinden kesmeleri inceleyeceğiz ve AVR mikrodenetleyicisinde dahili ve harici olmak

üzere kaç kesme olduğunu anlatacağız. Zamanlayıcılardan sonra kesmeleri bitirince geriye kalan konuların yükü oldukça azalacaktır. Kesme konusu zamanlayıcılar kadar zor olmasa da bilinmesi gereken başlıca konulardan ve mikrodenetleyicinin en temel birimlerinden olduğu için üzerinde durulması gerekir. Örneğin SPI ya da I2c birimini bilmeden yine pek çok iş yapılabilir ama kesmeleri bilmeden çoğu iş eksik kalır. Sonraki derste görüşmek üzere.

Kesme Adları ve Yazmaçları

Önceki derste kesmelere bir giriş yapmıştık. Kesme konusu çok uzun olmadığı için şimdi kesme adlarını ve yazmaçlarını vererek iç kesmeleri bitireceğiz. Ondan sonra konumuz dış kesmeler olacak. Şimdi kesme adlarını ve yazmaçlarını vererek iç kesmeleri bitirelim.

Kesme Adları

1	RESET	Reset kesmesi, harici ayak, açılışta, kararma resetinde ve WTC resette
2	INT0	Harici Kesme İsteği 0
3	INT1	Harici kesme İsteği 0
4	PCINT0	Ayak değişme kesme isteği 0
5	PCINT1	Ayak değişme kesme isteği 1
6	PCINT2	Ayak değişme kesme isteği 2
7	WDT	WDT Zamanlayıcısı Kesmesi
8	TIMER2_COMPA	Zamanlayıcı 2 Karşılaştırma Eşleşmesi A
9	TIMER2_COMPB	Zamanlayıcı 2 Karşılaştırma Eşleşmesi B
10	TIMER2_OVF	Zamanlayıcı 2 Taşma Kesmesi
11	TIMER1_CAPT	Zamanlayıcı 1 Yakalama Olayı
12	TIMER1_COMPA	Zamanlayıcı 1 Karşılaştırma Eşleşmesi A
13	TIMER1_COMPB	Zamanlayıcı 1 Karşılaşma Eşleşmesi B

14	**TIMER1_OVF**	**Zamanlayıcı 1 Taşma Kesmesi**
15	**TIMER0_COMPA**	**Zamanlayıcı 0 Karşılaştırma Eşleşmesi A**
16	**TIMER0_COMPB**	**Zamanlayıcı 1 Karşılaştırma Eşleşmesi B**
17	**TIMER0_OVF**	**Zamanlayıcı 1 Taşma**
18	**SPI STC**	**SPI Transfer Tamamlandı**
19	**USART_RX**	**USART Alım Tamamlandı**
20	**USART_UDRE**	**USART Veri Yazmacı Boş**
21	**USART_TX**	**USART Gönderim Tamamlandı**
22	**ADC**	**ADC Çevirim Tamamlandı**
23	**EE READY**	**EEPROM Hazır**
24	**ANALOG COMP**	**Analog Karşılaştırıcı**
25	**TWI**	**I2C Arayüzü**
26	**SPM READY**	**SPM Hazır**

MCUCR Yazmacı

Yazmacın Biti	7	6	5	4	3	2	1	0
Bit Adı	—BOŞ—	BODS	BODSE	PUD	—BOŞ—	—BOŞ—	IVSEL	IVCE
Erişim		Y/O	Y/O	Y/O			Y/O	Y/O

Bit 6 – BODS : BOD Uyku

BODS biti uyku sırasında BOD'u devre dışı bırakmak için bir (1) yapılmalıdır. Brown-Out Detection (Kararma Saptaması) mikrodenetleyiciye giden besleme geriliminin belli bir seviyeyi aşacak kadar kararması yani düşmesidir. Bu sayede yeterli gücü alamayan mikrodenetleyici kararsızlığa sebebiyet vermemek için kendini yeniden başlatır.

Bit 5 – BODSE : BOD Uyku Etkin

Bu bit BODS denetleme bitinin ayarını etkin hale getirir.

Bit 4 – PUD : Pull-UP devre dışı

Bu bir bir (1) yapıldığında temel giriş ve çıkış portlarının dahili pull-up özelliği devre dışı bırakılır.

Bit 1 – IVSEL : Kesme Vektör Seçimi

Eğer bu bir sıfır (0) yapılırsa kesme vektörleri flash program hafızasının başına yerleştirilir. Eğer bir (1) yapılırsa kesme vektörleri ön yükleyici bölümnün başına yüklenir.

Bit 0 – IVCE : Kesme Vektör Değişikliği Etkin

IVSEL bitindeki değişikliği uygulamak için IVCE biti bir (1) yapılmalıdır.

Buraya kadar kesmeler hakkında gereken bilgileri anlattık. Sonraki konumuz ise dış kesmeler olacaktır. Sonraki derste görüşmek üzere.

Dış Kesmeler

Önceki iki dersimizde kesmelere giriş yapmış ve iç kesmeleri anlatmıştık. Şimdi daha uzun bir konu olan dış kesmelere giriş yapacağız ve ardından kesme konusunu bitireceğiz.

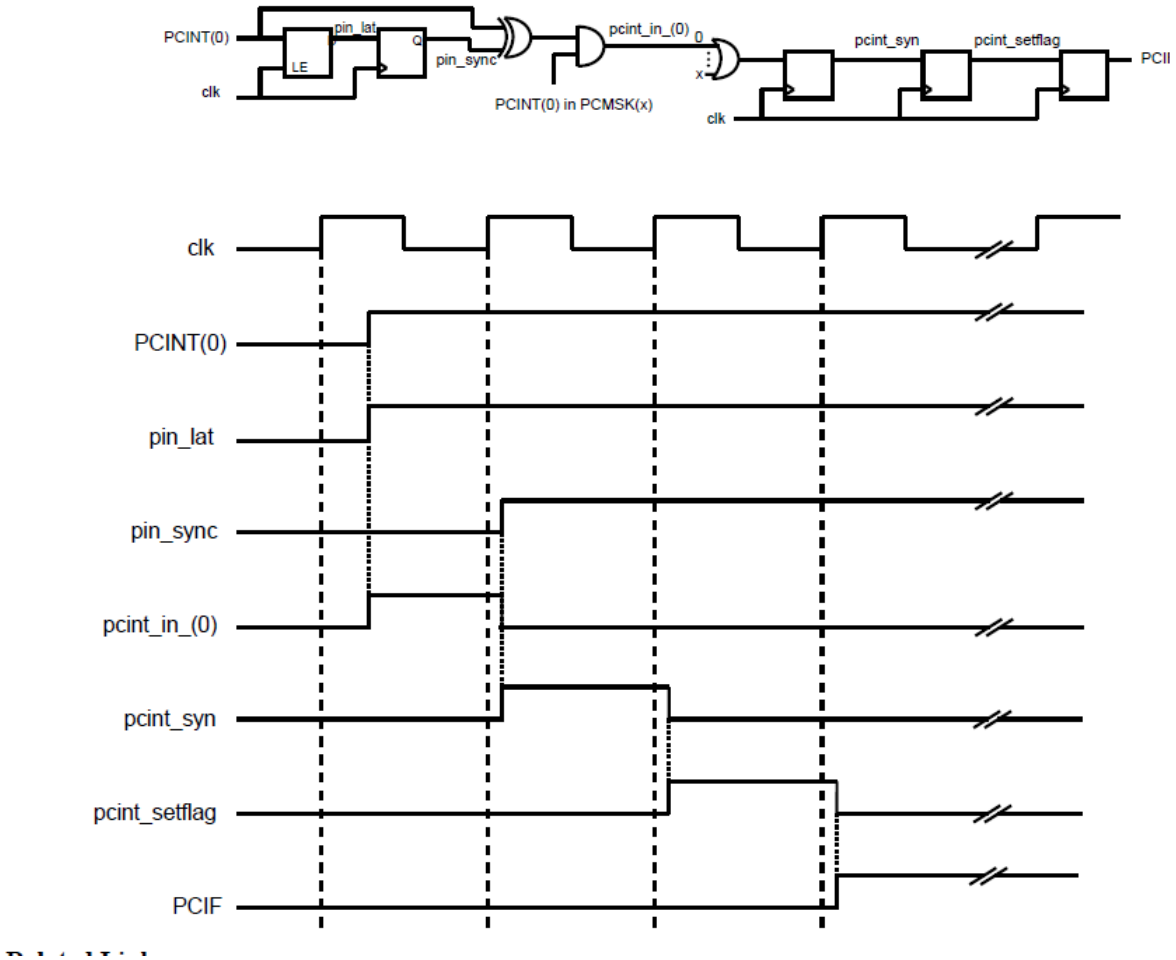
Dış kesmeler adından da anlaşılacağı üzere mikrodenetleyicinin iç birimlerinde gerçekleşmeyip dışarıdan bir uyaranla gerçekleşen kesmelere denir. Örneğin mikrodenetleyicinin ayağına bir düğme bağladığımızda bunun sebep olduğu kesme dış kesme iken mikrodenetleyicinin USART veri alışverişinde kullandığı kesmeler iç kesmedir. Kısacası dış kesme dediğimiz INT ya da PCINT ayaklarına uygulanan elektriğin sebep olduğu kesmedir. Böylece mikrodenetleyici kesmeler ile sadece kendi birimleri ile değil dış dünya ile de etkileşime geçebilir. Bu ayakları dersin başlarında verdiğimiz şemadan görebilirsiniz.

Ayak değişimi kesme talebi 2 (PCI2) PCINT[23:16] ayaklarından herhangi biri değişime uğrarsa yürütülür. Ayak değişimi kesme talebi 1 (PCI1) ise PCINT [14:8] ayaklarından herhangi biri değişime uğrarsa yürütülür. Ayak değişimi kesme talebi 0 (PCI0) ise PCINT[7:0] ayakları değişikliğe uğradığında yürütülür. PCINT ayaklarındaki ayak değişimi

kesmesi asenkron olarak algılanır. Bu yüzden bu kesmeleri uyku modundayken uyandırmak için kullanabiliriz.

Dış kesmeler düşen kenarda veya yükselen kenarda tetiklenebilir. Bu ayar dış kesme denetim yazmacı A (EICRA) yazmacından yapılır. Ayak değişimi kesme zamanlaması diyagramı aşağıdaki gibidir.

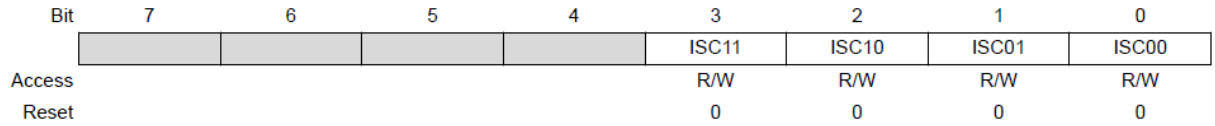
Figure 17-1. Timing of pin change interrupts



Resim: ATmega328P – Microchip Technology , sf. 88, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Erişim Tarihi: 25.08.2018

Yazmaçlar

EICRA – Dış Kesme Denetim Yazmacı A



Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 3:2 – ISC1n Kesme Algılama Denetimi [n=1:0]

Dış kesme 1'in INT1 ayağından yürütülmesi için SREG yazmacının I bayrak biti ve uygun kesme maskesi bir (1) konumunda olmalıdır. Seviye ve kenarlara göre INT1 kesmesinin yürütülme tablosu aşağıda verilmiştir. INT1 ayağının değeri kenarı tespit etmeden önce algılanır. Bu kesmenin yürütülmesi için bu ayağa giden sinyalin en azından bir saat çevirimi uzunluğunda olması gereklidir. Daha kısa uzunlukta olan sinyallerin bir garantisi yoktur. Eğer düşük seviye kesme seçildiyse düşük seviyenin kesmeyi yürütecek kadar uzun sürmesi gerekir.

Değer	Açıklama
00	INT1 LOW seviyesinde kesme yürütür
01	INT1'deki herhangi bir değişiklik kesme yürütür.
10	INT1'in düşen kenarı kesme yürütür.
11	INT1'in yükselen kenarı kesme yürütür.

Bit 1:0 – ISC0n : Kesme Algılama Denetimi 0 [n = 1:0]

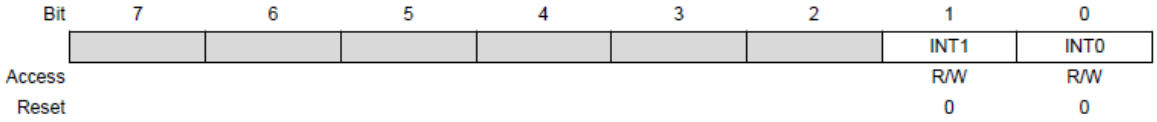
Dış kesme 0'ın INT1 ayağından yürütülmesi için SREG yazmacının I bayrak biti ve uygun kesme maskesi bir (1) konumunda olmalıdır.Seviye ve kenarlara göre INT0 kesmesinin yürütülme tablosu aşağıda verilmiştir. INT0 ayağının değeri kenarı tespit etmeden önce algılanır.

Değer	Açıklama
00	INT0 LOW seviyesinde kesme yürütür
01	INT0'daki herhangi bir değişiklik kesme yürütür.
10	INT0'ın düşen kenarı kesme yürütür.
11	INT0'in yükselen kenarı kesme yürütür.

Sonraki yazımızda dış kesme yazmaçlarına devam edeceğiz ve bu konuyu bitireceğiz.

Dış kesme yazmaçlarını anlatmaya bu yazımızda devam ediyoruz. Şimdi yazmaçlara kaldığımız yerden devam edelim.

EIMSK – Dış Kesme Maske Yazmacı



Bit	7	6	5	4	3	2	1	0
							INT1	INT0
Access							R/W	R/W
Reset							0	0

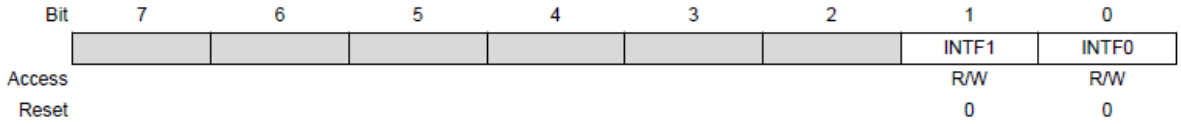
Bit 1 – INT1 : Dış kesme talebi 1 etkin

INT1 biti ve SREG yazmacındaki I biti bir (1) olduğunda harici ayak kesmesi etkin hale gelir. Dış kesme denetleme yazmacındaki (EICRA) kesme algılama denetim bitleri (ISC11 ve ISC10) bu kesmenin yükselen veya düşen kenarda ya da ayak durumuna göre olup olmayacağını kararlaştırır. INT1 ayağı çıkış olarak tanımlansa da bu ayağa uygulanan gerilim kesmeyi yürütecektir.

Bit 0 – INT0 : Dış kesme talebi 0 etkin

INT0 biti ve SREG yazmacındaki I biti bir (1) olduğunda harici ayak kesmesi etkin hale gelir. Dış kesme denetleme yazmacındaki (EICRA) kesme algılama denetim bitleri (ISC01 ve ISC00) bu kesmenin yükselen veya düşen kenarda ya da ayak durumuna göre olup olmayacağını kararlaştırır. INT0 ayağı çıkış olarak tanımlansa da bu ayağa uygulanan gerilim kesmeyi yürütecektir.

EIFR – Dış Kesme Bayrak Yazmacı



Bit	7	6	5	4	3	2	1	0
							INTF1	INTF0
Access							R/W	R/W
Reset							0	0

Bit 1 – INTF1 : Dış Kesme Bayrağı 1

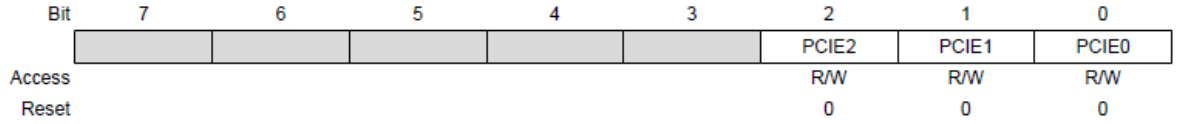
INT1 ayağındaki kenar veya mantık durumu değiştiğinde INT1 ayağı kesmeyi tetiklediğinde INTF1 biti bir (1) olur. Eğer SREG yazmacındaki I biti ve EIMSK yazmacındaki INT1 biti bir (1) yapılırsa işlemci kesme fonksiyonuna atlar. Bu bayrak kesme fonksiyonu yürütüldüğünde otomatik sıfırlanır. Ya da bu bayrağa bir (1) yazılarak sıfırlamak mümkündür. Bu bayrak INT1 kesmesi seviye kesmesi olarak ayarlandığında her zaman sıfır (0) konumundadır.

Bit 0 – INTF0 – Dış Kesme Bayrağı 0

INT0 ayağındaki kenar veya mantık durumu değiştiğinde INT0 ayağı kesmeyi tetiklediğinde INTF0 biti bir (1) olur. Eğer SREG yazmacındaki I biti ve EIMSK yazmacındaki INT0 biti bir (1) yapılırsa işlemci kesme fonksiyonuna atlar. Bu bayrak kesme fonksiyonu

yürütüldüğünde otomatik sıfırlanır. Ya da bu bayrağa bir (1) yazılarak sıfırlamak mümkündür. Bu bayrak INT0 kesmesi seviye kesmesi olarak ayarlandığında her zaman sıfır (0) konumundadır.

PCICR – Ayak Değişimi Kesmesi Denetleme Yazmacı



Bit 2 – PCIE2 : Ayak değişimi kesmesi Etkin 2

PCIE2 biti ve SREG yazmacındaki I biti bir (1) konumuna getirilirse ayak değişimi kesmesi 2 etkinleştirilir. PCINT[23:16] ayaklarındaki bir değişim bu kesmeyi yürütür. PCINT[23:16] ayakları tek tek PCMSK2 yazmacından etkinleştirilebilir.

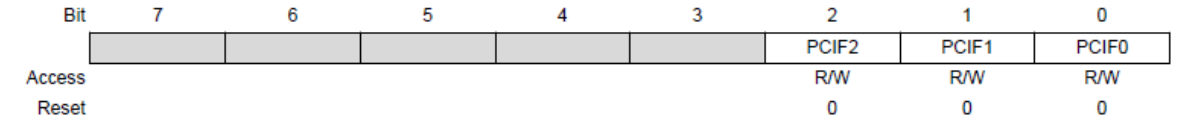
Bit 1 – PCIE1 : Ayak değişimi kesmesi Etkin 1

PCIE1 biti ve SREG yazmacındaki I biti bir (1) konumuna getirilirse ayak değişimi kesmesi 1 etkinleştirilir. PCINT[14:8] ayaklarındaki bir değişim bu kesmeyi yürütür. PCINT[14:8] ayakları tek tek PCMSK1 yazmacından etkinleştirilebilir.

Bit 0 – PCIE0 : Ayak değişimi kesmesi Etkin 0

PCIE0 biti ve SREG yazmacındaki I biti bir (1) konumuna getirilirse ayak değişimi kesmesi 0 etkinleştirilir. PCINT[7:0] ayaklarındaki bir değişim bu kesmeyi yürütür. PCINT[7:0] ayakları tek tek PCMSK0 yazmacından etkinleştirilebilir.

PCIFR – Ayak Değişimi Kesmesi Bayrak Yazmacı



Bit 2 – PCIF2 : Ayak Değişimi Kesme Bayrağı 2

PCINT [23:16] ayaklarından birinde mantıksal değişim olduğunda kesme talebi tetiklenir ve bu durumda PCIF2 biti bir (1) konumuna geçer.

Bit 1 – PCIF1 : Ayak Değişimi Kesme Bayrağı 1

PCINT [14:8] ayaklarından birinde mantıksal değişim olduğunda kesme talebi tetiklenir ve bu durumda PCIF1 biti bir (1) konumuna geçer.

Bit 0 – PCIF0 : Ayak Değişimi Kesme Bayrağı 0

PCINT [7:0] ayaklarından birinde mantıksal değişim olduğunda kesme talebi tetiklenir ve bu durumda PCIF0 biti bir (1) konumuna geçer.

PCMSK2 – Ayak Değişimi Maske Yazmacı 2

Bit	7	6	5	4	3	2	1	0
	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 0, 1 , 2, 4, 5, 6, 7 – PCINT 16, PCINT17, PCINT18, PCINT19, PCINT20, PCINT21, PCINT22, PCINT23 : Ayak değişimi Etkinleştirme Biti

Bu yazmaç hangi ayaklardan kesme gerçekleştireceğini belirler. Bu bitlerden biri ya da birkaçı ve PCICR yazmacındaki PCIE2 biti bir (1) yapılırsa uygun ayaktaki değişmeye göre kesme gerçekleşir. PCMSK2 yazmacındaki bitler sıfır (0) yapılırsa o kesme devre dışı bırakılır.

PCMSK1 – Ayak Değişimi Maske Yazmacı 1

Bit	7	6	5	4	3	2	1	0
		PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 0, 1, 2, 3, 4, 5, 6 – PCINT8, PCINT9, PCINT10, PCINT11, PCINT12, PCINT13, PCINT14 : Ayak Değişimi Etkinleştirme Biti

Bu yazmaç hangi ayaklardan kesme gerçekleştireceğini belirler. Bu bitlerden biri ya da birkaçı ve PCICR yazmacındaki PCIE1 biti bir (1) yapılırsa uygun ayaktaki değişmeye göre kesme gerçekleşir. PCMSK1 yazmacındaki bitler sıfır (0) yapılırsa o kesme devre dışı bırakılır.

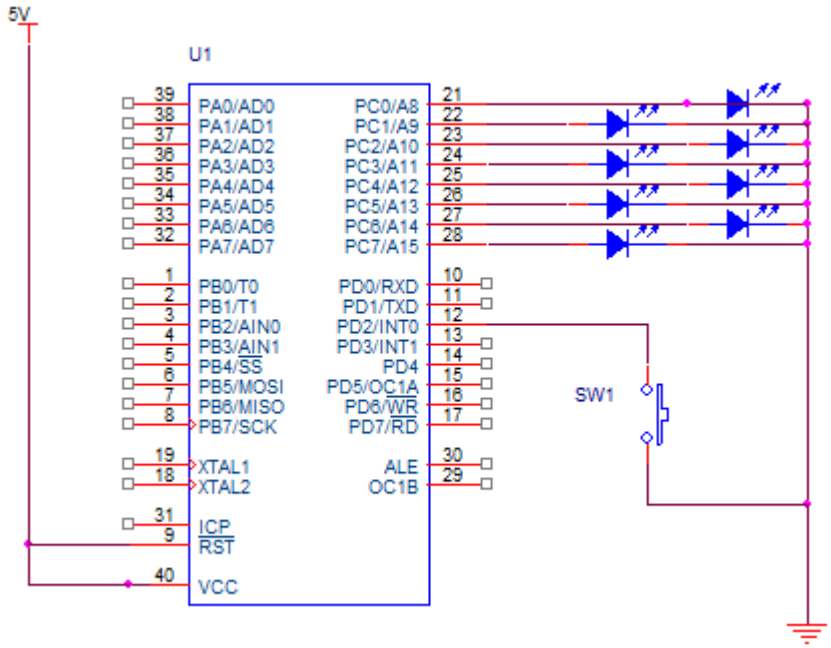
PCMSK0 – Ayak Değişimi Maske Yazmacı 0

Bit	7	6	5	4	3	2	1	0
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7:0 – PCINTn : Ayak Değişimi Etkinleştirme Biti [n = 7:0]

Bu yazmaç hangi ayaklardan kesme gerçekleştireceğini belirler. Bu bitlerden biri ya da birkaçı ve PCICR yazmacındaki PCIE0 biti bir (1) yapılırsa uygun ayaktaki değişmeye göre kesme gerçekleşir. PCMSK0 yazmacındaki bitler sıfır (0) yapılırsa o kesme devre dışı bırakılır.

Bütün dış kesme yazmaçlarını bitirdiğimize göre şimdi dış kesmelerin kullanıldığı bir örneği açıklayarak konumuzu bitirelim. Bu devrede dışarıdan bağlı bir düğme ve bir porta bağlı olan toplam 8 adet led bulunur. Düğmeye bastıkça bu ledler sıra ile yanacaktır. Programın en önemli özelliği ise bağlı olan düğmenin dış kesme ile kullanılmasıdır. Devremizin şeması aşağıdaki gibidir.



Resim : http://www.avr-tutorials.com/sites/default/files/INT0_0.png

Şimdi kodu vererek bu kodu satır satır anlatalım.

```
/*
 *  Yazar : AVR Tutorials
 *  Website: www.AVR-Tutorials.com
   Düzenleyen : Gökhan Dökmetaş
*/

#include <avr/io.h>
#include <avr/interrupt.h>

#define F_CPU 4000000UL
#include <util/delay.h>
 // Buraya kadar bildiğimiz kodlar. İşlemci 4MHz'de çalışacak.
#define DataPort        PORTC   // PORTC dataport olarak tanımlanıyor.
#define DataDDR         DDRC    // Port adlarına böyle isim verebiliriz.

//INT0 için kesme fonksiyonu. Kesme yürütüldüğünde işletilecek kodlar.
ISR(INT0_vect)
{
        unsigned char i, temp;

        _delay_ms(500); // Düğme arkını engellemek için bekleme fonksiyonu.

        temp = DataPort;        // DataPort'un değerini ver.

        /* Bu döngü dataporttaki ledleri beş kere yakar.*/
        for(i = 0; i<5; i++)
        {
                DataPort = 0x00;
                _delay_ms(500);// 500 ms bekle
                DataPort = 0xFF;
                _delay_ms(500);// 500 ms bekle
        }

        DataPort = temp;        //Dataporta eski değerini ver. (0)
}

int main(void)
{
        DDRD = 1<<PD2;          // ,INT0 kesmesi için kullanılan PD2 ayağı giriş
        PORTD = 1<<PD2;         // Dahili pull-up dirençlerini etkinleştir.

        DataDDR = 0xFF;         // Dataportu çıkış yap
        DataPort = 0x01;        // 1 değerini yükle.

        EIMSK = 1<<INT0;                // INT0 kesmesini etkinleştir
        EICRA = 1<<ISC01 | 1<<ISC00;    // Yükselen kenarda INT0 kesmesi yürüyecek

        sei();                          //Genel kesmeleri etkinleştir.

    while(1)
    {
                if(DataPort >= 0x80)
                        DataPort = 1;
                else
                        DataPort = DataPort << 1;       // Birer bit sola kaydır

                _delay_ms(500);// 500 mili saniye bekle
    }
```

Bu program kodlardan anlaşıldığı üzere normal çalışma döngüsünde devreye bağlı olan sekiz ledi sağdan sola sırayla 500 mili saniye aralıklarla yakacaktır. 0x80 değeri

0b10000000 değerine denk geldiği için 8. led yandığı zaman port tekrar 1 değerine çekilip ilk ledin yanması sağlanacaktır. Böylelikle bu kayan ışık programı sürekli olarak çalışacaktır. Bu sürekli çalışan programı kesmek için INT0 kesmesi kullanılmıştır. Bu kesmeyi yürütmek için PD2 ayağı giriş olarak tanımlanmış, buna bir düğme bağlanmış ve dahili pull-up özelliği etkinleştirilmiştir. Böylelikle düğmeye basıldığında kesme yürütülecektir. Dış kesmeler için kullanılan iki komut ve kesme fonksiyonu vardır. Geri kalan komutlar temel giriş ve çıkış özelliğinde olduğu için derslerin en başında anlatılmıştır.

ISR(INT0_vect) Bu kesme fonksiyonu INT0 kesmesi yürütüldüğünde işletilecek kodları içinde bulundurur.

GICR = 1<<INT0; GICR yazmacının INT0 biti bir (1) yapılarak INT0 kesmesi etkinleştirilir.

EICRA = 1<<ISC01 | 1<<ISC00; Kesmenin ne zaman yürütüleceği EICRA yazmacının bitleri ile kararlaştırılır.

Dış Kesme Vektör Adları

Kesmeleri kullanırken kesme fonksiyonunun içine vektör adı yazarak o fonksiyonun hangi kesme için yürütüleceğini belirliyorduk. avr/interrupt.h başlık dosyası ile kullanılan bu kesme fonksiyonlarının söz dizimleri şu şekildedir. **sei();** Kesme yürütmeyi etkin hale getirir. **cli();** Kesme yürütmeyi devre dışı bırakır. **ISR(vektör_adı) { komutlar; }**Yürütülecek kesme fonksiyonunu tanımlar. Şimdi vektörlere aşağıdaki tablodan göz atalım. Bu bizim için bir referans tablosu olacaktır.

Tablo Aşırı Derecede Uzun Olduğu İçin Kitaba Alınmamıştır. Lütfen Aşağıdaki Bağlantıdan Takip Ediniz.

http://www.lojikprob.com/avr/c-ile-avr-programlama-40-kesme-vektor-adlari/

EEPROM

EEPROM bellekler elektrik ile yazılıp silinebildiğinden üzerinde defalarca okuma ve yazma işlemi yapılabilir. Diğer eski ROM belleklerde bu bir defaya mahsus programlama veya ışık ile silme ve elektrikle programlama şeklinde olduğu için pek kullanışlı değildir. Çalışma ortamında entegre sadece kullanmaya mahsus olup üretim esnasında programlanır. ROM ve PROM bellekler bir Atari kaseti yapmak için uygun olsa da bizim yapacağımız projeler genellikle kontrol ve otomasyon devreleri olduğu için kullanıcının belirlediği ayar verileri ve diğer önemli verileri sürekli olarak okuyup yazmamız gerekecektir. EEPROMlar elektronik dünyasında oldukça yaygın olarak kullanılan hafıza birimleridir. Okuma ve yazma hızları biraz düşük kaldığı için günümüzde yüksek işlem gücü ve kapasite gerektiren

uygulamalarda FLASH bellekler kullanılsa da 8-bit seviyesinde en uygun hafıza birimleri EEPROMlardır.

ROM, PROM, EPROM ve EEPROM olarak zamanla gelişen bu hafıza üniteleri RAM belleklerden oldukça farklı bir yapıya sahiptir. ROM bellek kalıcı bellek olmasından dolayı aynı zamanda fiziksel bellek özelliği taşır. RAM bellekte bilgi sadece elektrikten ibaret iken ROM bellekte farklı statüye geçip fiziksel olarak kaydedilir. Bu fiziksel kayıt aynı zamanda elektrikle de okunup yazılabilir özelliktedir. Kısacası EEPROM bellek aynı bir flash bellek gibi kullanıcı verilerinin kalıcı olarak depolandığı hafıza birimidir. Uygulama esnasında kaydedilen bu veriler programdan bağımsızdır.

ATmega328P entegresinde 1KB EEPROM bulunur ve bu EEPROMu programlamak ve okumak için özel EEPROM yazmaçları vardır. Bu yazmaçların belli bitlerine belli verileri yazarak okuma işlemini yaparız. Bizim için iki önemli değer vardır. Birincisi adres ikincisi ise veridir. EEPROMlarda her veri 8 bitlik adreslere bağlı veri hücrelerinde tutulur. Bu veri hücrelerindeki veriyi okumak için adres bilgisi şarttır. Bu adresleri aynı bir çizgili defterin satırları gibi düşünebiliriz. Bu satırlar defterde numaralı olmasa da burada numaralandırılmış ve 0, 1, 2, 3 vd. olarak devam etmektedir. Her satırda ise 8 bitlik yani 0-255 arası bir sayı değeri veya harf saklanabilir. Bu saklanan veri yıllar boyu üzerine bir veri yazılmadıkça orada bulunur. Cihaz yıllar boyu çalışmasa dahi silinmez. Program hafızası FLASH bellek olup EEPROMdan daha üstün bir teknolojide olsa da uygulama esnasında program hafızası ROM bellek olarak çalışır. Yani sadece program esnasında yazılıp sonradan yazılıp silinemez. Elimizde sadece okunabilir kalıcı bellek olarak program hafızası bulunduğu için mikrodenetleyicilerde EEPROM bellekleri genellikle kullanıcı belleği olarak kullanırız.

Yarı iletken üreticileri harici entegre olarak EEPROM bellekleri de üretmektedir. Bu bellekler SPI, I2C, Seri veya Paralel iletişim protokolleriyle mikroişlemci ve mikrodenetleyiciler tarafından kullanılabilmektedir. ATmega mikrodenetleyicilerde dahili EEPROM bulunması bizi ekstra bir masraftan kurtaracaktır.

EEPROMlar hakkında aşırı derece teknik bilgiye ihtiyacımız yoktur. Çünkü prensibi ve kullanması oldukça basittir. EEPROM üretmeyeceğimiz için bu kadarını bilmemiz yeterli olur. Bir sonraki derste her zaman olduğu gibi yazmaçları anlatarak konumuza devam edeceğiz ve sonrasında örnek kodları inceleyeceğiz.

EEPROM Yazmaçları

Derslere yazmaya başlarken herhangi bir “İçindekiler” listesi veya şablon kullanmadım. Tamamen doğaçlama yürütülen bu dersler zaman ile kendi orjinal formatına sahip oldu. Böyle bir formata sahip olmasını asla kalitesizlik göstergesi olarak görmemek gereklidir. Çünkü diğer yayınlara bakıp onları esas alarak yürüttüğüm bir çalışma özgünlük açısından daha zayıf olacaktı. Çalışmayı diğer ders yazılarındaki seviyede tutup bilginin tamamını vermeyip en basit seviyeden anlatmakla yürütseydim yazması da anlaşılması da daha kolay olacaktı. Biz kolay olmasını, herkese hitap etmesini, anlaşılır olmasını değil faydalı olmasını gözettiğimiz için verilebilecek faydanın azami seviyede olmasına özen gösteriyoruz. Bu yüzden çoğu hususdan feragat ederek bu çalışmayı yürütmekteyiz.

Fark edilirse günümüzde basılan bilişim ve elektronik alanındaki kitapların neredeyse tamamı giriş seviyesinde olmaktadır. İleri seviyede veya alt bir konuda bir kitap pek görülmemektedir. Bu, bu alandaki literatürün en büyük eksikliği olmaktadır. Bir okur aynı konuda onlarca kitap okusa dahi giriş seviyesinde kalıyorsa bu kitapların faydasının sorgulanması gereklidir. Öte taraftan giriş seviyesinde olmayan kaynaklar yeterli ilgiyi görmemektedir. Bu hazırladığımız kaynak hiçbir zaman hak ettiği ilgiyi görmeyecek olsa da biz görevimizi yerine getireceğiz.

AVR derslerini sadece AVR konuları üzerinden yürütmemin bir sebebi var. Bu dersler bittikten sonra dijital elektronik, mikroişlemci mimarisi ve gömülü sistemlerde C dili dersleri gibi başka ders yazısı dizisi yürütmeyi düşünüyorum. AVR dersleri hemen bittikten sonra ise Arduino kaynak kodu incelemesi yapacağımız bir yazı dizimiz olacak. Böylelikle Arduino'dan AVR'ye geçenler aynı zamanda Arduino'nun nasıl yazıldığını da öğrenecek. Arduino'nun kaynak kodunu öğrenerek Arduino fonksiyonlarını ve kütüphanelerini rahatça AVR'ye uyarlayarak kullanabileceksiniz. Bu durumda Arduino'nun hiçbir artısı kalmış olmayacak. Bu yüzden AVR derslerinden sonra yürütülmesi gereken en önemli çalışmanın bu olacağını düşünüyorum.

Şimdi dersimize başlayalım,

Teknik veri kitapçığında EEPROMlar hafıza birimleri üst başlığı altında anlatılmaktadır. Burada tüm konuları anlatma gibi bir gayemiz olsa da Flash ve SRAM gibi birimleri mikroişlemci mimarisini anlattığımız zamanlarda AVR Mimarisi alt başlığında anlatacağız. C dilinde programlama için AVR mimarisini bu seviyede bilmek gerekli değildir. Yine de C dilinde programlama yaparken EEPROM bellekler üzerinde çalışmak diğer alanlara göre biraz daha alt seviyededir, diyebiliriz. Bunu dememizin sebebi burada sadece veri ve kontrol yazmaçları yer almayıp bir de bunun yanında adres bilgisi de yer almaktadır.

Buradan anlaşılacağı üzere denetim, Veri ve Adres yazmacı olmak üzere üç ayrı yazmaç türü vardır. Şimdi bu yazmaçları tek tek açıklayalım.

EEARH – EEPROM Adres Yazmacı (Üst)

Bit	7	6	5	4	3	2	1	0
							EEAR9	EEAR8
Access							R/W	R/W
Reset							x	x

Adres yazmacının üst ve alt olmak üzere iki ayrı yazmaçtan oluşup toplamda 10-bit olmasının sebebi tek bir yazmacın 8-bit olup toplamda 255 baytlık bir veriyi adresleyebilmesinden dolayıdır. ADC konusunda da gördüğümüz üzere 1024 farklı değeri saklayabilmek için 10 bitlik bir alana ihtiyaç vardır. 1024 baytlık bir EEPROM verisini adreslememiz gerektiği için 10-bitlik bir alana ihtiyacımız vardır. Bu yazmaç ise üst iki biti saklamak için kullanılır.

EEARL – EEPROM Adres Yazmacı (Alt)

Bit	7	6	5	4	3	2	1	0
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Bu yazmaç 10 bitlik adres verisinin alt 8 bitini saklamak için kullanılır. EEPROMdan bir veri okuyup yazmadan önce adres değerini bu yazmaca yazmak gereklidir. EEAR bitleri adres verisi bitleridir.

EEDR – EEPROM Veri Yazmacı

Bit	7	6	5	4	3	2	1	0
	EEDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bu yazmaç EEAR yani EEPROM Adres Yazmacının gösterdiği adresteki EEPROM verisini tutan yazmaçtır. Okuma ve yazma işlemleri bu yazmaç üzerinden yapılır. Her adresten 8 bitlik bir değer okunduğu için 16-bit integer, float ve karakter dizisi verilerini okurken adres üzerinde işlem yapmak gereklidir. Tek bir adresten tek bir harf veya bayt verisi okunabilir. EEDR bitleri EEPROM verisini saklayan bitlerdir.

EECR – EEPROM Denetim Yazmacı

Bit	7	6	5	4	3	2	1	0
			EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			x	x	0	0	x	0

Yukarıda adres ve veri yazmaçlarını anlattık. Geriye ise denetim yazmacı kaldı. Denetim yazmacı tek bir işleve sahip olmadığı için her bir bitinin ne işe yaradığını ezberlemenizi istemiyoruz. Fakat gerektiği zaman bu yazdığımız referans kaynağa müracaat edebilmeniz gerekli. Önemli olan bunu ezberlemek değil ne işe yaradığını ve ne yapılabildiğini öğrenmenizdir.

Bit 5:4 – EEPMn EEPROM Programlama Modu Bitleri [n = 1:0]

EEPE biti tetiklendiği zaman hangi programlama modunda işlem yapılacağı bu bitler ile belirlenir. Bu işlemler okuma ve yazma olarak iki ayrı işlem olsa da atomik işlem adı verilen silme ve yazmanın beraber yürütüldüğü bir işlem de vardır. EEPE biti bir (1) konumunda olduğu zaman EEPM bitlerine yapılan yazım işlemi görmezden gelinir. Aşağıdaki tabloda EEPM bitlerinin işlevi verilmiştir.

EEPM[1:0]	Programlama Süresi	İşlem
00	3.4 ms	Silme ve Yazma (Tek İşlemde)
01	1.8 ms	Silme
10	1.8 ms	Yazma
11	–	Rezerve (Kullanım Dışı)

Bit 3 – EERIE : EEPROM Hazır Kesmesi Etkin

Bu biti bir (1) yapmak EEPROM'un hazır olduğunda yürütülecek kesmeyi yürür hale getirir. Burada bu kesmenin yürütülmesi için SREG'deki I bitiyle genel kesmelerin etkin olması gereklidir. Kesmenin nasıl kullanılacağına dair bilgi edinmek için kesmeleri konu aldığımız önceki derslerimize bakabilirsiniz.

Bit 2 – EEMPE : EEPROM Ana Yazma Etkin

Bu bit, EEPE bitine bir (1) yazıldığında yazma işleminin yapılıp yapılmayacağını belirleyen bir güvenlik anahtarıdır. EEMPE bir (1) olduğu zaman EEPE 4 saat çevrimi içinde bir (1) olursa seçili adrese veri yazılır. Eğer EEMPE sıfır ise EEPE bitini bir (1) yapmanın bir etkisi yoktur. EEMPE biti bir (1) yapıldıktan sonra donanım dört saat çevrimi içerisinde bu biti tekrar sıfır (0) konumuna getirir.

Bir 1 – EEPE : EEPROM Yazma Etkin

Adres ve veri yazmaçları hazır olduğu zaman yazma işleminin gerçekleşmesi için EEPE bitinin bir (1) yapılması gereklidir. EEPE biti bir (1) yapılmadan önce EEMPE bitinin bir (1) yapılması gereklidir. Bu işlem bu sırada yürütülmezse EEPROM yazma işlemi gecikmeye uğrar. Aşağıdaki işlem sırasına göre EEPROM yazma işlemi yürütülmelidir.

- EEPE bitinin sıfır olmasını bekle.
- SPMCSR yazmacındaki SPMEN bitinin sıfır olmasını bekle.
- EEAR yazmacına yeni EEPROM adresini yaz.
- EEDR yazmacına yeni EEPROM verisini yaz.
- EEMPE yazmacına EEPE bitine sıfır (0) yazma esnasında bir (1) yaz.
- EEMPE yazmacına bir (1) yazmanın ardından dört saat çevrimi içerisinde EEPE yazmacına bir (1) yaz.

Bit 0 – EERE : EEPROM Okuma Etkin

Bu bit sayesinde EEPROMdaki belli bir adresten bir veriyi okuyabiliriz. EEAR yazmacına doğru adres yazıldıktan sonra EERE biti bir (1) yapılmalıdır. Böylelikle okuma işlemi tetiklenmiş olur. EEPROM okuma işlemi sadece bir işlemci komutunda yürütülür ve istenilen veriyi anında verir. EEPROM okunduğunda işlemci dört saat çevrimi süresince durdurulur ve sonraki işlemci komutu işletilir. EEPROM okuma işlemi 26368 saat çeviriminde ve 3.3ms zamanda gerçekleşmiş olur.

Fark edilirse bir baytı okumak için 3 ms kadar bir zaman gereklidir. Bu süre bir veya birkaç bayt seviyesinde görmezden gelinecek kadar az olsa da belleğin ciddi bir bölümünü kaplayan verileri okumak için katlanarak artıp uzun bir bekleme süresi haline gelecektir. Daha hızlı veri okumak için SD Kart veya Flash Bellek kullanmak gerekebilir. Çoğu zaman ihtiyaç duymasak da bunu bilmenizde fayda vardır.

Bu derste bütün EEPROM yazmaçlarını anlatmayı bitirdik ve sıra örnek kodları anlatmaya geldi. Bunu da bir sonraki derste anlatacağız ve konuyu bitireceğiz.

EEPROM Örnek Kod İncelemesi

AVR Programlarken çoğu zaman dahili bir kütüphane olmasa da EEPROM kütüphanesi derleyicinin içinde gelmektedir. Aslında bu kütüphaneyi kullanıp bütün bu yazmaçlardan ve bitlerden kurtulmamız mümkün olsa da biz bu alanda literatür oluşturacak seviyede bir ders dizisi yazma gayretinde olduğumuz için işi mümkün olduğunca alt seviyede tutmaya çalışacağız. Böylelikle anlaşılmadık bir nokta kalmayacak. Şimdi teknik veri kitapçığında verilen örnek EEPROM okuma ve yazma kodlarını satır satır inceleyelim.

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
/* Önceki okuma ve yazmanın bitmesini bekle */
while(EECR & (1<<EEPE))
;
/* Adres değerini güncelle */
EEAR = uiAddress;
/* Okuma işlemini başlat. */
EECR |= (1<<EERE);
/* Veri yazmacını değer olarak döndür */
return EEDR;
}
```

Okuma işlemini anlatmakla işe başlayalım. Okuma işlemi oldukça basit olup adresteki değeri döndürmekten ibarettir. Bu adres verisi **unsigned int uiAddress** olarak programcı tarafından fonksiyon çağrılırken yazılır.

while(EECR & (1<<EEPE)) EECR Yazmacındaki EEPE biti bir olduğu sürece program sonsuz döngüye sokulur. EEPE biti eğer bir okuma veya yazma işlemi yürütülüyorsa bir (1) konumunda olup işlem bitmesinin ardından donanım tarafından sıfır (0) konumuna çekilir. Yeni bir işlemin yürütülmesi için öncelikle önceki işlemin bitmesi gereklidir. Bu yüzden bu kontrol döngüsünü en başa alıyoruz.

EEAR = uiAddress; Aynı kodun Assembly dilindeki halini incelediğimde EEARH ve EEARL yazmaçlarına ayrı ayrı değer atandığını gördüm. Fakat C dilinde programlama yaparken sadece EEAR yazmacında değer atamak yeterli oluyor. Normalde üst ve alt yazmaçlara ayrı ayrı değer atanması gerekse de burada böyle bir kolaylık getirilmiştir.

EECR |= (1<<EERE); EERE biti bir (1) yapılarak okuma işlemi yürütülür. Yazmaçları açıkladığımız önceki derste bu bitin okuma işlemini başlatmakla görevli olduğunu açıklamıştık.

return EEDR; EEPROM Veri Yazmacı değer olarak döndürülür. Bu yazmaçta okuma işlemi bittikten sonra belirtilen adresteki okunan veri yer alır.

Sırada ise yazma işleminin yürütüldüğü bir fonksiyon var. Bunu da yukarıda olduğu gibi satır satır inceleyelim.

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
/* Önceki yazma işleminin bitmesini bekle*/
while(EECR & (1<<EEPE))
;
/* Adres ve Veri yazmaçlarını ayarla*/
EEAR = uiAddress;
EEDR = ucData;
/*EEMPE bitini bir yap */
EECR |= (1<<EEMPE);
/* EEPE bitini bir yaparak yazmaya başlat.*/
EECR |= (1<<EEPE);
}
```

Görüldüğü gibi bu fonksiyon **void** ile başlıyor. Çünkü yazma işleminde herhangi bir değer okumayacağız. Öncelikle 10 bitlik adres verisini **unsigned int uiAddress** olarak, yazılacak veriyi ise **unsigned char ucData** diye argüman olarak alıyoruz.

while(EECR & (1<<EEPE)) ; EECR Yazmacındaki EEPE biti bir olduğu sürece program sonsuz döngüye sokulur. EEPE biti eğer bir okuma veya yazma işlemi yürütülüyorsa bir (1) konumunda olup işlem bitmesinin ardından donanım tarafından sıfır (0) konumuna çekilir. Yeni bir işlemin yürütülmesi için öncelikle önceki işlemin bitmesi gereklidir. Bu yüzden bu kontrol döngüsünü en başa alıyoruz.

EEAR = uiAddress; Programcı tarafından belirlenen 10 bitlik adres verisi EEAR yani EEPROM Adres Yazmacına yazılır. Yazma işleminden önce muhakkak doğru adresin yazılması gereklidir. Bu adresi ise programcı çakışmadan belirlemek zorundadır. Hangi verinin hangi adreste bulunacağı keyfimize bağlıdır. Önemli olan bu değerlerin üst üste gelip birbiri üzerine yazılmamasıdır.

EEDR = ucData; char olarak bu veri yazılsa da bizim bir baytlık veri olarak anlamamız gereklidir. Bu değerlerin muhakkak işaretsiz (unsigned) olması gerektiğini söylememize gerek yoktur.

EECR |= (1<<EEMPE); EEPROMa yazma işlemini yapmadan önce bu güvenlik bitini bir yapmamız gereklidir. Kontağı açıp marşa basmak gibi bir işlem olduğu için önce kontağı açmamız gereklidir. Bu görevi bu bit yerine getirmektedir.

EECR |= (1<<EEPE); Bu bit ile yazma işlemini yürütmeye başlarız. Bütün bu hazırlıktan sonra son noktayı bu bite bir (1) yazarak koyarız ve işlem yürütülür.

Bir sonraki dersimizde EEPROM kütüphane fonksiyonlarını açıklayacağız ve konuyu bitireceğiz.

EEPROM Kütüphanesi

EEPROM konusunu var olan bir kütüphaneyi açıklayarak bitireceğiz. Sadece kütüphane açıklayarak da konuyu bitirebilsek de bu insanı hazırcılığa itecektir. AVR programlarken çoğu kütüphaneyi kendimiz yazmamız, var olan kütüphaneleri de kendimiz düzenlememiz gereklidir. Böylelikle kütüphaneye göre program değil programa göre kütüphane ortaya çıkacaktır. Bu alanda profesyonel bir iş ortaya koymak için bu şarttır.

AVR GCC derleyicisinde hazır olarak eeprom.h başlık dosyası vardır. Dışarıdan bir kütüphane indirmemize gerek yoktur ve Atmel Studio'yu yüklediğimiz andan itibaren bu kullanılabilir. Yine **include** direktifi ile bu başlık dosyasını programa dahil etmemiz gereklidir. Aşağıdaki kodu programın başında kullanarak EEPROM kütüphanesini programımıza dahil ederiz.

```
#include <avr/eeprom.h>
```

Makro açıklaması ile devam ederken burada atladığımız makroların olduğunu söylememizde fayda var. Bu makrolar IAR AVR derleyicisine uyumluluk için tanımlanmış makrolar olup bizim işimize yaramamaktadır. Yazının sonunda kaynak olarak belirttiğimiz bağlantıdan kütüphanenin tamamını inceleyebilirsiniz.

eeprom_busy_wait()

Bu makro EEPROMda okuma ve yazma işleri yürütülürken işlemciyi döngüye sokarak beklemeye alır. Örnek kodlarda belirttiğimiz beklemeyi bu fonksiyonla sağlarız.

eeprom_is_ready()

EEPROM yeni bir okuma ve yazma işine hazır olduğunda bu fonksiyon bir (1) değerini geri döndürür. Bu fonksiyonun nasıl yazılacağını ise yazmaçları ve örnek kodları incelediğimiz için biliyoruz. Yani istesek kendi EEPROM kütüphanemizi de yazabiliriz.

```
void eeprom_read_block(void * __dst, const void *__src, size_t __n)
```

Bu fonksiyon EEPROMdan bir veri bloğunu okuyup SRAM belleğine aktarır. Veri bloğunun boyutu __n ile belirlenir. __n verisine yazılan değer kadar bayt okuması yapılır. __src ise EEPROM adresi olup SRAM ise okunan değerin aktarılacağı adrestir. Veri blokları genellikle karakter dizileri olduğu için aktarım için dizi değişken tanımlanması gerekir. Örnek bir blok okuması aşağıdaki gibi olmalıdır.

```
#include <avr/eeprom.h>
void main(void)
{
 uint8_t StringVerisi[10];
 eeprom_read_block((void*)&StringVerisi, (const void*)20, 10);
}
```

Önce aktarılacak değerin adres değeri, sonra eeprom adresi (burada 20 olarak belirtilmiş 0-1023 arası olacak.) ve devamında uzunluk sabiti belirtilir. Bu fonksiyon 10 baytlık veriyi 20. adresten başlayarak StringVerisi dizisinin adresine sırayla kopyalar.

```
uint8_t eeprom_read_byte( const uint8_t * __p)
```
Bu fonksiyon EEPROMdan bir bayt okur ve bunu 8 bitlik değer olarak döndürür. __p değişkeni okunacak adres verisini bulundurur. Burada “*” işareti dikkatinizi çekmiştir. Bu işaretçi olarak alınan bir argüman değerini gösterir. Bunun için fonksiyon çağırılırken değerleri çevirmek gereklidir. Örnek kodda bu gösterilmiştir. Örneğin 50. adresteki bayt değerini okuyalım.

```
#include <avr/eeprom.h>
void main(void)
{
 uint8_t okunandeger;
 okunandeger = eeprom_read_byte((uint8_t*)50);
}
uint16_t eeprom_read_word( const uint16_t * __p)
```
Bu fonksiyon 16 bitlik word tipinde değişken okumayı sağlar. Yukarıda olduğu gibi bu da işaretçi ile belirlendiği için argüman olarak göndereceğimiz sabitin işaretçiye dönüştürülmesi gereklidir. Aşağıdaki örnek kodda 50. adresteki word değişkeni okunmaktadır.

```
#include <avr/eeprom.h>
void main(void)
{
 uint16_t wordverisi;
 wordverisi = eeprom_read_word((uint16_t*)50);
}
```
Yukarıdaki örneklerde olduğu gibi aşağıdaki fonksiyonların da bu şekilde kullanılması gereklidir.

```
uint32_t eeprom_read_dword( const uint32_t * __p)
```
Bu foksiyon 32-bit double word cinsinde değeri okur.

```
float eeprom_read_float( const float * __p)
```
Bu fonksiyon float cinsi değer okur.

```
void eeprom_write_byte( uint8_t * __p, uint8_t __value)
```
Bu fonksiyon byte cinsinde değer yazar.

```
void eeprom_write_word( uint16_t * __p, uint16_t __value)
```
Bu fonksiyon word cinsinde değer yazar.

```
void eeprom_write_dword( uint32_t * __p, uint32_t __value)
```
Bu fonksiyon double word cinsinde değer yazar.

```
void eeprom_write_float( float * __p, float __value)
```
Bu fonksiyon float cinsinde değer yazar.

```
void eeprom_write_block( const void * __src, void * __dst, size_t __n)
```
Bu fonksiyon blok değeri yazmak için kullanılır.

Okuma ve Yazma işleminden başka bir de Update (Güncelleme) özelliği vardır. Bu özelliğin tek getirisi mevcut değer aynı ise tekrar yazma işleminin yürütülmemesidir. Böylelikle EEPROMun ömrü uzatılmış olur. Bu özelliğin kullanıldığı fonksiyonların prototipleri aşağıdaki gibidir.

Bayt için,

```
void eeprom_update_byte( uint8_t * __p, uint8_t __value)
```

Word için,

```
void eeprom_update_word( uint16_t * __p, uint16_t __value)
```

Double word için,

```
void eeprom_update_dword( uint32_t * __p, uint32_t __value)
```

Float için,

```
void eeprom_update_float( float * __p, float __value)
```

Blok için,

```
void eeprom_update_block( const void * __src, void * __dst, size_t __n)
```

Bu fonksiyonların nasıl kullanılacağını örneklerle yukarıda açıkladığımız için tek tek açıklamaya gerek yoktur. Bu konuda anlatılacak pek bir şey kalmadığı için burada bitirelim ve yeni konuya geçelim.

Analog Karşılaştırıcı (Analog Comparator)

45. derse başlarken konulara şöyle bir baktığımızda anlatılacak konuların oldukça azaldığını görüyoruz. Şu ana kadar önemli konuların çoğunu bitirmemizin yanında toplam konu sayısı olarak konuların çoğunu anlatmış olduk. Geriye kalan konular ise zor konular olmayıp sizi yormayacaktır. Dersleri yazarken mümkün olduğu kadar birinci kaynakları (üretici tarafından yayınlanan dokümanlar) kullandığımız için bu kaynaklarda bahsedilmeyen bilgileri mümkün olduğu kadar AVR programlama derslerimize almayacağız. Bu ders dizisi bittikten sonra farklı bir başlık altında diğer konulardan bahsedeceğiz.

Şimdi yeni konumuz olan Analog Karşılaştırıcı birimine giriş yapalım.

Analog Karşılaştırıcı Birimi

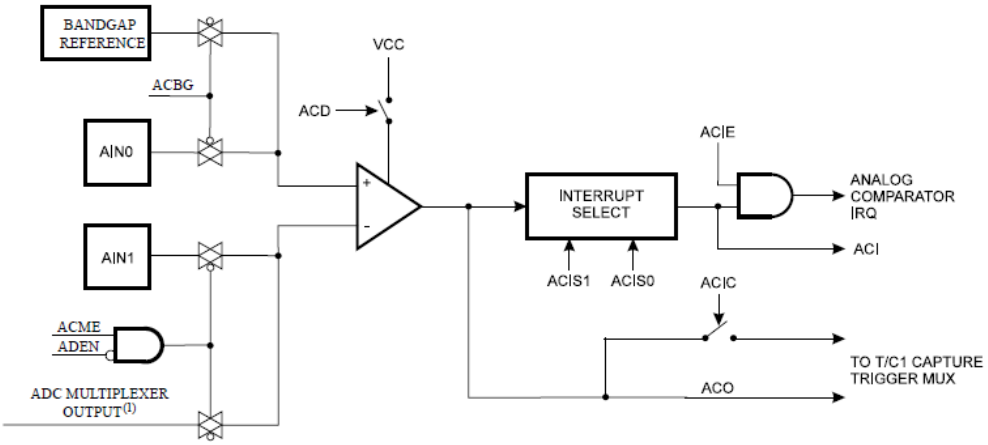
Analog karşılaştırıcı iki değeri analog olarak karşılaştırır ve bu iki değerin durumuna göre dijital bir çıkış verir. Aslında bu işlemi biz iki ADC kanalından okuma yapıp bunları if-else yapısı ile karşılaştırarak yapabiliriz. Fakat bu biraz usulsüz olacağı için bu görev için

mevcut olan birimi kullanmamız daha doğru olacaktır. Bir de Analog Karşılaştırıcı biriminde ayrı kesmeler ve özellikler vardır. O yüzden öğrenmeden geçmemek gereklidir.

Analog karşılaştırıcı pozitif ve negatif ayak olmak üzere iki adet ayağa sahiptir. Pozitif ayak AIN0 olarak, nagatif ayak ise AIN1 olarak adlandırılır. AIN0 ayağına giden gerilim AIN1 ayağına giden gerilimden yüksek ise analog karşılaştırıcının çıkışı (ACO) bir (1) konumuna gelir. Karşılaştırıcının çıkışı TC1 zamanlayıcısında giriş yakalama işlemi için de kullanılabilir. Bu özelliğin olması bu çıkışın sayma işleminde kullanılmasına olanak sağlar. Ayrıca karşılaştırıcı ayrı bir kesme yürütür. Analog karşılaştırıcı için özel bir kesmenin olmasının bize faydası dokunacaktır.

Analog karşılaştırıcıda ADC Multiplexer kullanmak için PRR.PRADC bitinin sıfır (0) yapılması gereklidir. Şimdi fikir edinmeniz açısından analog karşılaştırıcının blok diyagramını aşağıda verelim.

Figure 27-1. Analog Comparator Block Diagram



Analog karşılaştırıcı mikroişlemciden ayrı bir birim olup mikroişlemciden bağımsız olarak çalışır. Bütün analog ölçme ve karşılaştırma işlemini kendi içinde yürütür. Yürüttüğü kesmelerle mikroişlemciye müdahale eder. Mikrodenetleyicinin erişimi ise çıkış biti olan ACO bitini okumakla olur.

Analog Karşılaştırıcı Çoklu Giriş Ayağı

ADC[7:0] yani analog giriş ayakları analog karşılaştırıcının negatif ayağı yerine kullanılabilir. Bu özelliğin kullanılması için ADC kapalı konumda olmalıdır. ADC Denetim ve Durum Yazmacı B'deki Analog Karşılaştırıcı Çoklayıcısı Etkin hale getirilirse (ADCSRB.ACME) ve ADC kapatılırsa (ADCSRA.ADEN=0) ADMUX yazmacının son üç bitindeki çoklayıcı bitleri analog karşılaştırıcının negatif girişini belirler. Aşağıdaki tablodan daha anlaşılır halde görebilirsiniz.

ACME	ADEN	MUX[2:0]	Analog Karşılaştırıcı Negatif Girişi
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Şimdi yazmaçlara geçelim ve Analog Karşılaştırıcı hakkında anlatılmadık bir şey kalmasın.

ADCSRB – ADC Denetim ve Durum Yazmacı B

Bu yazmacın kullanımını daha önce açıkladığımız için şu bağlantıdan tekrar okuyun. http://www.lojikprob.com/avr/c-ile-avr-programlama-16-analog-dijital-cevirici-adc-yazmaclari/

ACSR – Analog Karşılaştırıcı Denetim ve Durum Yazmacı

Bit	7	6	5	4	3	2	1	0
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
Access	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bu yazmaç analog karşılaştırıcı hakkında bütün işlemleri yapacağımız yazmaçtır.

Bit 7 – ACD : Analog Karşılaştırıcı Devre Dışı

Bu bit bir (1) yapıldığında analog karşılaştırıcıya giden besleme devre dışı bırakılır. Bu bit istenilen her zaman devre dışı bırakmak için kullanılabilir. Güç tasarrufu için aklımızda bulundurmakta fayda vardır. ACD biti değiştirilirken ACIE biti ile kesmenin de devre dışı bırakılması gereklidir.

Bit 6 – ACBG : Analog Karşılaştırıcı Bant Aralığı Seçimi

Bu bit bir (1) yapıldığında bant aralığı referansı pozitif girişin yerine geçer. Bu bit sıfır (0) olduğunda ise AIN0 pozitif giriş olarak kullanılır.

Bit 5 – ACO : Analog Karşılaştırıcı Çıkışı

Analog karşılaştırıcının çıkışını bu bitten okuruz. Arada senkronizasyon işlemi olduğu için 1-2 saat çevirimi gecikmeli okunur.

Bit 4 – ACI: Analog Karşılaştırıcı Kesme Bayrak Biti

Bu bit donanım tarafından analog karşılaştırıcı kesmesi yürütüldüğünde bir (1) konumuna getirilir. ACI biti kesme fonksiyonu yürütüldükten ya da üzerine bir (1) yazıldıktan sonra sıfırlanır.

Bit 3 – ACIE : Analog Karşılaştırıcı Kesmesi Etkin

Analog karşılaştırıcının kesme yürütmesini istiyorsak bu biti bir (1) yapmamız gereklidir.

Bit 2 – ACIC : Analog Karşılaştırıcı Giriş Yakalama Etkin

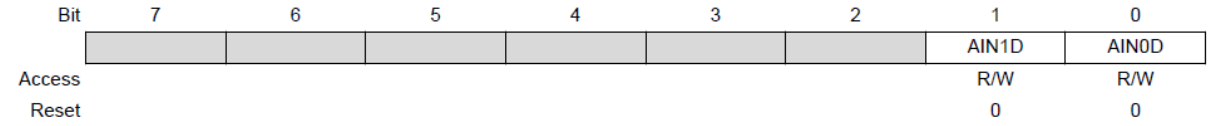
Bu bit bir (1) yapıldığında TC1 zamanlayıcısının giriş yakalama fonksiyonu analog karşılaştırıcı tarafından tetiklenir. Bu bit sıfır (0) yapıldığında analog karşılaştırıcı ile zamanlayıcı arasında bir bağlantı kalmaz.

Bit 1:0 – Analog Karşılaştırıcı Kesme Modu Seçimi [n = 1:0]

Bu bitlerin durumuna göre analog karşılaştırıcı kesme modu seçimi yapılır. Kesme modlarını aşağıdaki tabloda görebilirsiniz.

ACIS1	ACIS0	Kesme Modu
0	0	Çıkış Değişmesinde Kesme
0	1	Rezerve (Kullanım Dışı)
1	0	Düşen çıkış kenarında kesme
1	1	Yükselen çıkış kenarında kesme

DIDR1 – Dijital Girişi Devre Dışı Bırakma Yazmacı



Bu yazmaçtaki AIN1D ve AIN0D bitleri bir yapılırsa bu ayaklardaki dijital giriş devre dışı bırakılır. Böylelikle analog sinyal uygulanan ayaklarda dijital giriş tamponu devre dışı bırakılarak güç tasarrufu sağlanır.

Böylelikle analog karşılaştırıcı hakkında anlatılabilecek her konuyu anlatmış olduk. Konu oldukça basit olduğu için örnek kod vermemize gerek yoktur. Bir sonraki derste yeni bir konuya geçeceğiz.

SPI (Serial Peripheral Interface) İletişim Protokolü

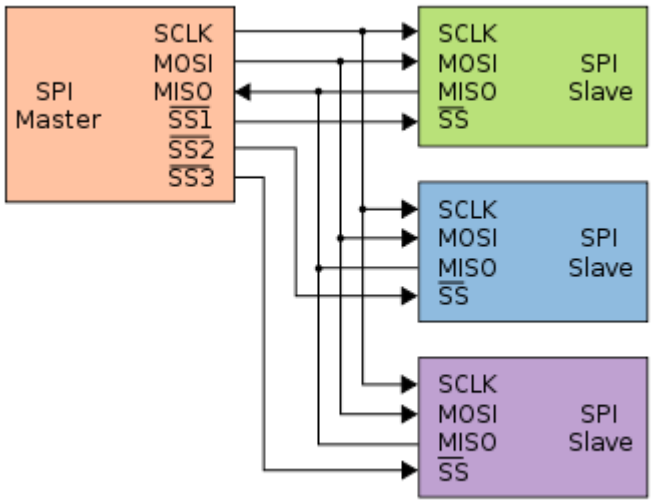
Türkçe olarak Seri Çevrelik Arayüz adını verebileceğimiz bu iletişim protokolü senkron olarak işleyen bir seri iletişim protokolüdür. Aynı hat üzerinde sayısız aygıt ile yüksek hızda seri iletişime geçebiliriz. Aygıt seçmek için ne bir adres ne bir programa ihtiyaç vardır. Aygıtlarda bulunan seçme ayağının dijital konumunda yapacağımız değişiklik ile bu seçme işlemini yaparız. Bu yönden diğer iletişim protokollerine (özellikle I2C) göre üstünlüğü bulunur. Her ne kadar sayısız cihaz bağlayabiliriz desek de dijital giriş ve çıkış ayağımız kadar bir sınırlamamız vardır. Bu sınırlamayı kaldırmak için çoklayıcı entegreler gibi dış birimleri kullanmamız gerekebilir. Bu iletişim protokolü Motorola firması tarafından gömülü sistemler üzerinde kullanmak üzere tasarlanmıştır. SPI protokolü herhangi bir amaca veya donanıma yönelik bir sistem olmadığı için oldukça geniş bir alanda kullanılmaktadır. Pek çok mikrodenetleyicide SPI protokolü bulunmakla beraber SPI üzerinden çalışan entegre ve birimler belli bir markaya ait değildir. Aynı zamanda da belli bir alanda da gruplaşmamıştır. Ada dilinin havacılıkta kullanılması, CAN Bus iletişiminin otomotiv sektöründe kullanılması gibi bazı dil, platform ve protokoller belli bir sektörde kullanılmak üzere tasarlanmıştır. Bunlar gibi olmayan SPI protokolünü ise gömülü sistemlerin hemen her alanında çalışacak programcıların öğrenmesi gereklidir.

SPI iletişiminde veri sinyali gönderilirken aynı zamanda da bir saat sinyali gönderilir. UART protokolünde gördüğümüz üzere baud oranı ile sınırlandırılmış bir çalışma sahamız yoktur. SPI iletişiminde ana aygıt (master) ve uydu aygıt (slave) olmak üzere aygıtlar ikiye ayrılır. Tüm uydu aygıtlar tek bir ana aygıta bağlanmak zorundadır. Bu özellik protokolün temellerinden olup bizim ana aygıtımız ise mikrodenetleyici olacaktır. Mikrodenetleyiciye bağlayacağımız tüm aygıtlar uydu görevinde olup okuma ve yazma işini ana aygıt yapacaktır. Bu iletişimde her zaman ana aygıtın sözü geçmektedir. SPI protokolünü kullanırken mikrodenetleyici neredeyse tamamen ana aygıt olarak kullanılmaktadır. Eğer birden fazla mikrodenetleyiciyi SPI ile birbirine bağlamak istiyorsak uydu aygıt olarak da kullanma imkanımız vardır.

SPI protokolünde dört adet ayak kullanılır. Bu ayaklardan üçü ortak olup biri ise her aygıt için ayrı olmalıdır. Bu ayakların açıklaması aşağıdaki gibidir.

- SCLK – Saat Sinyal Ayağı Olup Tüm Aygıtlarda Ortaktır. Bu ayaktan çıkan saat sinyali senkronizasyonu sağlar.
- MOSI – “Master Out Slave In” kelimelerinin kısaltımı olup ana aygıttan uydu aygıtlara giden veri bu ayak üzerinden iletilir.
- MISO – “Master In Slave Out” kelimelerinin kısaltımı olup uydu aygıtlardan ana aygıta giden veri bu ayak üzerinden iletilir.
- SS – “Slave Select” kelimelerinin kısaltımı olup her uydu ayak için ayrı bir seçilmiş ayaktır. Bu ayaklar hangi uydu aygıt üzerinde okuma ve yazma işlemi yapacağımızı belirler.
  Şimdi SPI biriminin teknik veri kitapçığında belirtilen başlıca özelliklerine göz atalım.

- Tam dubleks, üç ayaktan senkronize veri transferi
- Ana ya da Uydu çalışma (İstenilirse ana bir aygıta bağlanabilir.)
- LSB (Son bit önde) ya da MSB (Baş bit önde) Veri Aktarımı
- Yedi Programlanabilir Bit Oranı
- İletişim Bittiğinde Kesme Bayrağı
- Yazma Çakışması Koruma Bayrağı
- Bekleme modundan uyanış
- Çift Hızlı (CK/2) Ana SPI Modu
  Örnek bir SPI devresinin şeması aşağıdaki gibi olmalıdır.



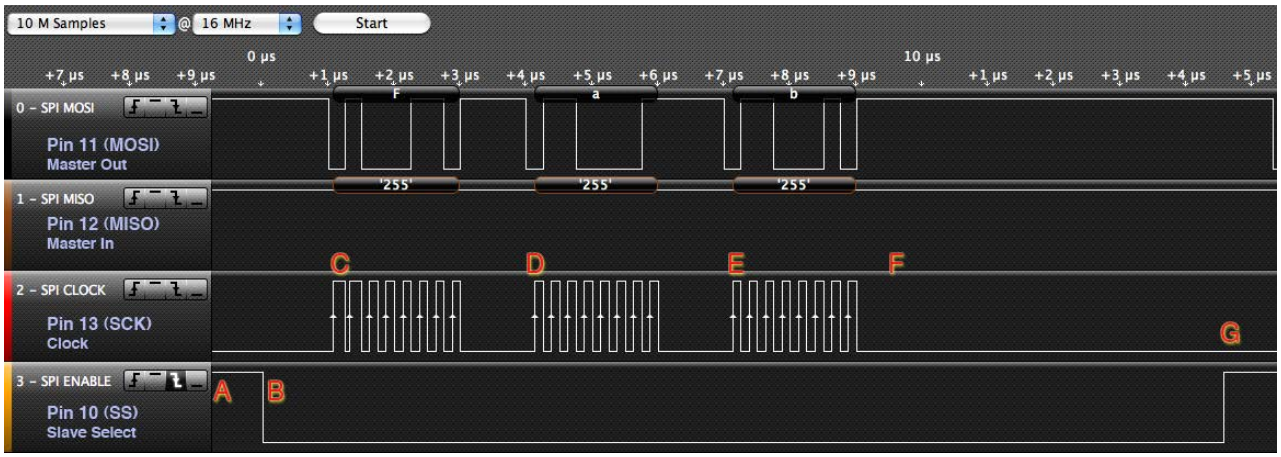
Resim:
https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI_three_slaves.svg/350px-SPI_three_slaves.svg.png

Resimde görüldüğü üzere SCLK, MOSI ve MISO ayakları ortak SS ayakları ise ana aygıtın ayrı ayaklarıdır. Biz de SPI bağlantımızı buna göre yapmak zorundayız.

SPI ile kullanabileceğimiz aygıt sayısı burada sayılamayacak kadar fazladır. Başlıca aygıtlar TFT ekranlar, algılayıcılar, hafıza birimleri, sd kart, modüller olarak sıralansa da onlarca firma tarafından onlarca farklı alanda kullanılmak üzere sayılamayacak kadar çok aygıt üretilmektedir. Bu aygıtları AVR mikrodenetleyiciler ile kullanabilmek için ise SPI öğrenmek şarttır. Bir sonraki yazıda SPI protokolünü ayrıntılı olarak anlatmaya devam edeceğiz.

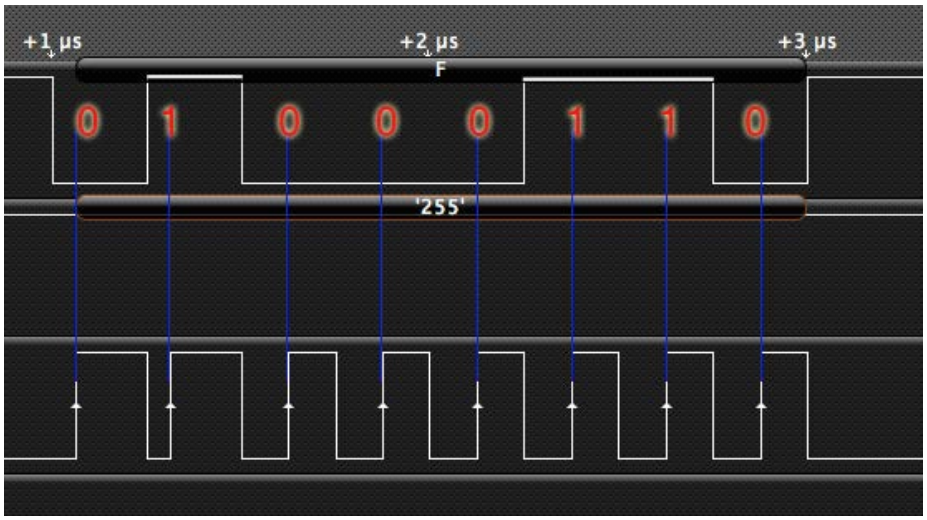
SPI Protokolünü AVR’de Kullanmak

AVR ile ilgili konulara geçmeden önce örnek bir SPI iletişiminin lojik analizör grafiğini verip bunun analizini yapalım. Böylelikle SPI protokolünün nasıl çalıştığını kolaylıkla anlayacaksınız.



Resim : http://www.gammon.com.au/images/SPI_logical_analyzer_1.png Burada grafiği anlamamız için bazı noktalar harfler ile işaretlendirilmiştir. Bu harfleri madde madde açıklayalım.

- A : SPI iletişim olmadığı zaman ayakların bu vaziyette olması gereklidir. SS ayağı HIGH konumdadır ve diğer ayaklar durgundur.
- B : Bu noktada öncelikle uydu aygıt üzerinde okuma ve yazma işlemi yapmak için uydu aygıtın iletişimde etkin hale getirilmesi gereklidir. Bu da SS ayağının LOW konuma çekilmesiyle olur. Uydu aygıt bu ayağın LOW konuma çekilmesiyle iletişimin başlayacağını anlamış olur.
- C : İlk bayt verisi uydu aygıta gönderilir. Her 8 bit için 8 ayrı saat çevrimi yapılmaktadır. MOSI ayağı gönderilecek veriye göre LOW ya da HIGH konuma geçer. MOSI ayağı UART protokolünde olduğu gibi çalışsa da burada SCK ile saat sinyali de gönderilmektedir. Saat sinyali ile veri sinyalinin nasıl karşılıklı olduğuna dikkat ediniz.
- D – Diğer bayt gönderilir.
- E – Diğer bayt gönderilir.
- F – Veri gönderimi yok fakat SS etkin. Uydu aygıt veri gönderimine açık.
- G – SS HIGH konuma getirilerek devre dışı bırakılır ve veri gönderme süreci bitirilir. Böylelikle seçili uydu aygıt SPI sinyallerini görmezden gelir.
  Eğer uydu aygıt bir veri yollayacaksa bu süreçte MISO ayağından veri yollayabilirdi. Şimdi tek bir karakterin gönderimi sırasında olan mantıksal grafiğe bakalım.



Resim: http://www.gammon.com.au/images/SPI_logical_analyzer_2.png

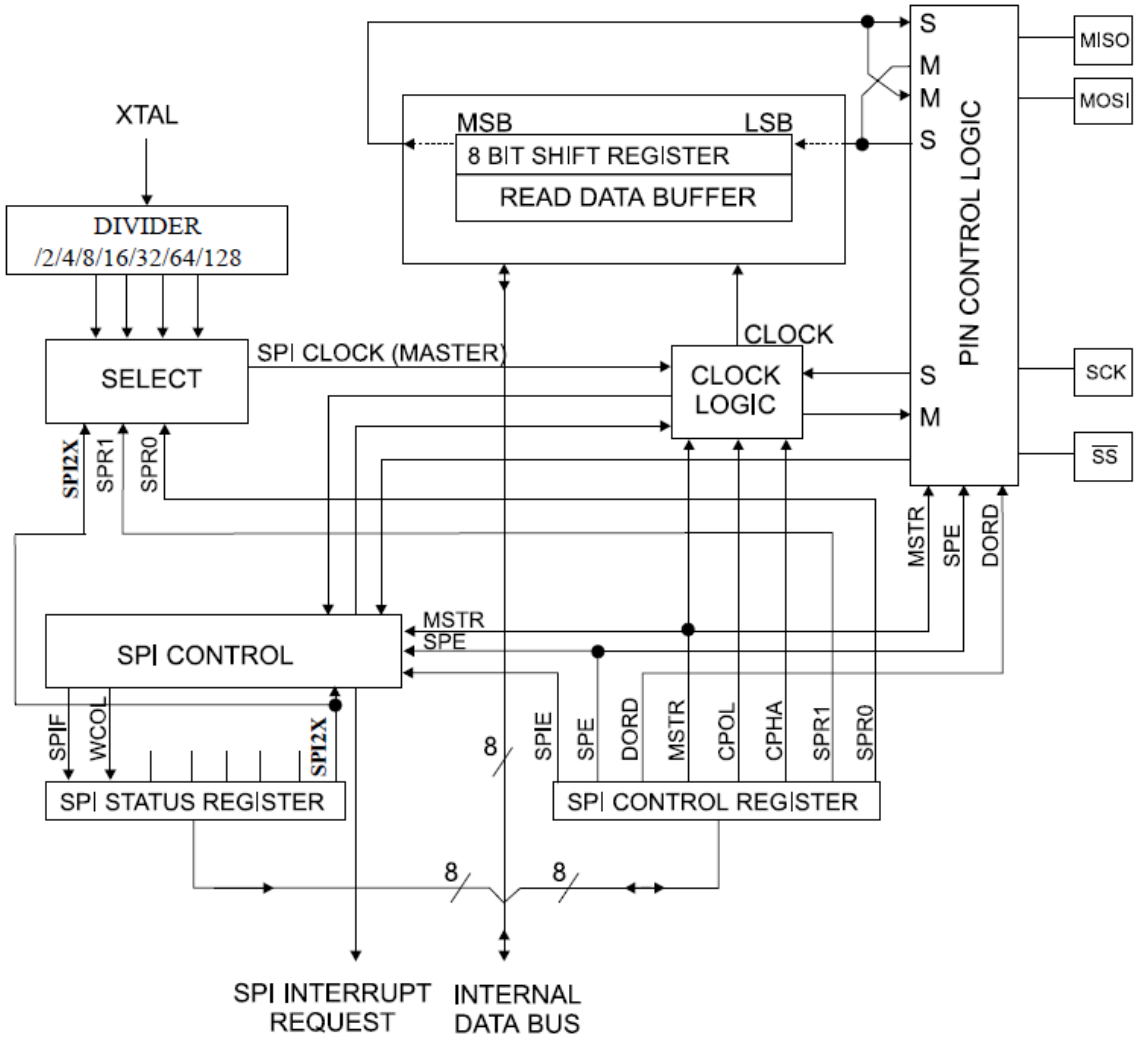
Burada F harfi (0b01000110) gönderilirken her saat sinyalinin yükselen kenarında veri yollanmaktadır. Bitler tek tek sırayla yollanırken baş bit önde son bit ise sonra yollanmaktadır. Yollanma moduna göre bu tersten de olabilir. Ayrıca yine moda göre düşen kenarda vd. konumlarda kullanabiliriz. Bu konuları şimdi açıklayacağız.

Hız

Atmega328P entegresinde yukarıda görüldüğü üzere 3 mikrosaniyede bir bayt gönderilmektedir. Bu da 325.5 kilobayt / saniye hıza denk gelmektedir. Ayrıca teorik olarak 888 KB/s hıza kadar ulaşmak mümkündür bu da yakın bir zamana kadar pek çoğumuzun kullandığı internetten daha hızlı bir hız anlamına gelmektedir.

SPI Hakkında temel bilgiler ve Arduino ile kullanımı için şu bağlantıyı ziyaret edebilirsiniz, http://www.gammon.com.au/spi

AVR mikrodenetleyicilerde SPI biriminin blok diyagramı aşağıdaki gibidir.



Bu blok diyagramı UART protokolünü okuduysanız biraz tanıdık gelecektir. Her birimde olduğu gibi durum, denetim ve veri yazmaçları yer alsa da burada bir de belli bir saat sinyali ve ön derecelendirici vardır. Artık bu noktadan sonra mikrodenetleyici birimlerinin mantığını anladığımız için yazmaçların görevlerini az çok öngörmemiz lazımdır.

SPI ana aygıtı iletişimi uydu seçimi (SS) ayaklarını sıfır (0) yaparak başlatır. Ana ve uydu aygıt kendi kayan yazmaçlarından (shift register) gönderilecek veriyi hazırlar. Ana aygıt, SCK ayağından gereken saat sinyalini üretir. Veri her zaman ana aygıttan uydu aygıtlara MOSI ayağından gönderilir. Uydu aygıtlardan ana aygıtlara ise MISO ayağından gönderilir. Her veri paketinden sonra ana aygıt SS ayağını bir (1) yaparak uydu aygıtla senkronize olur.

SS ayağı eğer otomatik bir denetime sahip değilse her zaman yazılım tarafından denetlenmesi gerekir. SS ayağı denetlendikten sonra SPI Veri yazmacına uygun veri yazılmalıdır. Veri yazıldıktan sonra SPI saat üreteci çalışmaya başlar ve donanım sekiz bitlik veriyi uydu aygıta doğru kaydırır yani seri olarak gönderir. Eğer SPI kesmesi etkin ise kesme yürütülür. Ana aygıt tekrar bir veri göndermek istiyorsa veri yazmacına o veriyi yazar. Eğer veri alışverişi bitti ise SS ayağı bir (1) konumuna getirilir.

Eğer mikrodenetleyici uydu aygıt olarak tanımlanırsa SS ayağı sıfır (0) olana kadar SPI birimi uykuya geçer. Bu durumda yazılım SPI Veri Yazmacını güncelleyebilir fakat bu verinin gönderilmesi için muhakkak SS ayağının ana aygıt tarafından sıfır (0) konumuna getirilmesi gerekir. Bir baytın gönderimi tamamlanınca gönderim bittiğine dair bayrak biti bir (1) konumuna geçer. Eğer SPI kesmesi etkin hale getirilmişse kesme yürütülür. Uydu aygıt ana aygıta göndereceği veriyi tekrar veri yazmacına yazabilir.

Sonraki dersimizde yazmaçları ve geri kalan teorik konuları anlatarak SPI konusuna devam edeceğiz.

SPI Birimi Yazmaçları

Bu dersimizde her zaman olduğu gibi yazmaçları anlatarak konumuza devam edeceğiz. Kopyala yapıştır kodla, ezbere kod yazma ile ne kadar üst seviye dillerde program yazılabilse de biz gömülü sistemlerde çalıştığımız için üst seviye bilgiye ihtiyacımız vardır. Üst seviye bilgi alt seviyede çalışmayı gerektirir. Burada alt seviyeden kastımız ise donanıma en yakın seviye demektir.

Şimdi yazmaçları açıklayalım ve sonraki dersimizde ise örnek kodları inceleyerek konumuza devam edelim.

SPCR0 – SPI Denetim Yazmacı 0

Bit	7	6	5	4	3	2	1	0
	SPIE0	SPE0	DORD0	MSTR0	CPOL0	CPHA0	SPR01	SPR00
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – SPIE0 : SPI0 Kesme Etkin

Bu bit bir (1) yapıldığında ve SPIF biti de bir (1) yapıldığında SPI kesmesini yürütür.

Bit 6 – SPE0 : SPI0 Etkin

Bu bit bir (1) yapıldığında SPI etkinleştirilir. Herhangi bir SPI işlemini gerçekleştirmek için bu bit bir (1) yapılmalıdır.

Bit 5 – DORD0 : Veri0 Düzeni

DORD biti bir (1) yapıldığında alt bit öncelikli olarak veri gönderilir. DORD biti sıfır (0) yapıldığında ise üst bit öncelikli olarak veri gönderilir.

Bit 4 – MSTR0 : Ana/Uydu 0 Seçimi

Bu bit bir (1) yapıldığında SPI iletişimini ana aygıt modunda çalıştırır. Eğer sıfır (0) yapılırsa mikrodenetleyici uydu modunda çalışır. Eğer SS ayağı giriş olarak tanımlanır ve LOW konuma çekilirse MSTR biti sıfırlanır ve SPIF biti bir (1) konumuna gelir. Yani uydu moduna otomatik olarak geçer. Tekrar ana modu açmak için bu bitin bir (1) yapılması gerekir.

Bit 3 – CPOL0 : Saat0 Kutbu

Bu bit bir (1) yapılırsa bekleme konumunda SCK ayağı HIGH konumda kalır. Eğer sıfır (0) yapılırsa bekleme konumunda LOW konumda kalır. Bu SPI modunu seçmek için kullanılır.

Bit 2 – CPHA0 : Clock0 Evresi

Bu bitin durumuna göre örneklenecek veri düşen kenarda veya yükselen kenarda alınır. Sıfır (0) iken yükselen kenarda bir (1) iken ise düşen kenarda veri örneklenir. İleride SPI modlarına değineceğimiz için bu iki bitin toplamda 4 ayrı SPI modunu seçtiğini bilmemiz yeterlidir.

Bit 1:0 – SPR0n : SPI0 Saat Oranı Seçimi n [n = 1:0]

Bu bit ana aygıtın SCK bacağının hızını belirlemeye yarar. Uydu aygıtta bu bitler çalışmaz. Osilatör ve SCK ayağı arasındaki bağlantı aşağıdaki tabloda verilmiştir.

SPI2X	SPR01	SPR00	SCK Frekansı
0	0	0	Osilatör / 4
0	0	1	Osilatör / 16
0	1	0	Osilatör / 64
0	1	1	Osilatör / 128
1	0	0	Osilatör / 2
1	0	1	Osilatör / 8
1	1	0	Osilatör / 32
1	1	1	Osilatör / 64

SPSR0 – SPI Durum Yazmacı 0

Bit	7	6	5	4	3	2	1	0
	SPIF0	WCOL0						SPI2X0
Access	R	R						R/W
Reset	0	0						0

Bit 7 – SPIF0 : SPI Kesme Bayrak Biti

Seri gönderim tamamlandığında SPIF bayrak biti bir (1) konumuna geçer. Eğer SPIE yani SPI Kesme Biti bir (1) konumunda ise kesme yürütülür. Eğer SS ayağı giriş olarak tanımlanmış ve LOW konumuna gelmiş ise bu bit yine bir (1) konumuna geçer. SPIF biti kesme fonksiyonu yürütüldükten sonra donanım tarafından sıfırlanır.

Bit 6 – WCOL0 : Yazma Çakışması Bayrak Biti

Eğer veri gönderimi sırasında SPI Veri Yazmacına bir yazma işlemi yapılırsa bu bit bir (1) konumuna geçer. Bu bit bir (1) konumuna geçtikten sonra ilk okumanın gerçekleşmesiyle tekrar sıfırlanır.

Bit 0 – SPI2X0 : SPI Çift Hız Biti

Bu bir bir yapıldığında SPI hızı (SCK Frekansı) ikiye katlanır. Sadece ana aygıt modunda çalışmaktadır.

SPDR0 – SPI Veri Yazmacı 0

Bit	7	6	5	4	3	2	1	0
	SPID[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Bu yazmaç hem gönderilen veriyi hem de alınan veriyi içerisinde barındırır. USART protokolünde gördüğümüz üzere her defasında 8 bitlik bir veri gönderilir. SPI yazmaçları çok da fazla değildir. Toplamda 3 adet yazmaç bulunur ve bütün iletişim bu yazmaçların denetimi ve okuyup yazılması ile sağlanır.

Yazmaçları anlatırken biraz ezberci anlatmak durumunda kalıyoruz. Yapısı gereği daha alt seviye bir ünite olmadığı için “bu bu işe yarar” demekten başka bir seçeneğimiz yok. Çünkü daha ilerisi ancak mikrodenetleyici üretmek için gerekli bilgilerden oluşuyor. Üreticinin de bu bilgileri bizimle paylaşması beklenemez. Yine de bu dersler bittikten sonra mikroişlemci mimarisi derslerimizde kendi işlemcimizi üretme yolunda teorik bilgileri sizinle paylaşacağız.

Bir sonraki derste örnek kodları inceleyeceğiz ve eksik kalan yerleri tamamlayacağız.

SPI Örnek Kod İncelemesi

SPI Protokolünü kullanmak için öncelikle açıp hazır hale getirmemiz gereklidir. Aynı ADC veya UART birimlerinde olduğu gibi bir “initialize” yani hazır hale gelme süreci gereklidir.

Bunu ise program başında şu fonksiyon ile yapıyoruz. İleride Arduino kodlarını inceleyeceğimiz için SPI.begin() fonksiyonunun da buna benzerlik gösterdiğini görebilirsiniz.

```
void SPI_MasterInit(void)
{
/* MOSI ve SCK Çıkış olarak tanımla, diğerleri giriş olacak*/
DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
/* SPI'ı ana aygıt modunda başlat, saat oranını belirle*/
SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}
```

void SPI_MasterInit(void) Bu fonksiyon ne argüman alır ne de bir değer döndürür. O yüzden iki kısım da void olarak tanımlandı.

DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK); Burada hiç görmediğimiz adlar var. Bu adlar C diline entegre edilen ve mikrodenetleyicinin SPI ayaklarının giriş ve çıkış (DDR) adreslerini bulunduran sabitlerdir. DDR_SPI SPI portunun adresi (örneğin PORTB) DD_MISO, DD_MOSI, DD_SCK ise bu ayakların adresidir (PD1, PD2 gibi..). Bu sadece kullanımı kolaylaştıran bir yapı olup programlayıcı hatalarının önüne geçer. Mikrodenetleyicinin kendisinde böyle bir yapı yoktur. Burada MOSI ve SCK ayakları yapısı itibariyle çıkış olarak tanımlanmıştır. Tanımlanmayan ayaklar ise giriştir.

SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); SPCR yani SPI denetim yazmacında SPE biti bir yapılır. Bu bitin bir (1) yapılması ile SPI biriminin etkinleştirildiğini önceki derste anlattık. MSTR bitinin bir (1) yapılmasıyla ile de SPI ana aygıt modunda çalışmaya başlar. SPR0 biti bir (1) yapılarak SPI saat hızının 16'da biri hızda çalışır.

SPI birimini başlatmak için gereken kodlar bu kadardı. Hangi ayakta hangi şekilde başlatmamız gerekiyorsa bunu yazmaçlardaki bitlerle oynayarak sizin yapmanız gereklidir. Bu kod standart bir modda SPI birimini başlatır. Bazen SPI aygıtlarla farklı modda iletişime geçmemiz gerekebilir. Bu kodu ezbere yazarak bunu yapmamız mümkün değildir. Önceki derste anlattığımız üzere CPOL ve CPHA bitlerinin konumuna göre SPI çalışma modları değişiyordu. İki bit olduğu için toplamda dört farklı çalışma modumuz vardı. Bu modları özetlersek,

- Mod 0 (standart), saat normalde LOW konumunda (CPOL = 0) yükselen kenarda veri örneklenir. (CPHA = 0)
- Mod 1, saat normalde LOW konumunda (CPOL = 0) düşen kenarda veri örneklenir. (CPHA = 1)
- Mod 2, saat normalde HIGH konumunda (CPOL = 1), düşen kenarda veri örneklenir. (CPHA = 0)
- Mod 3, saat normalde HIGH konumunda (CPOL = 1), yükselen kenarda veri örneklenir. (CPHA = 1)

SPI üzerinde çalışma yaparken bu modların olduğunu unutmamak gereklidir. Şimdi örnek bir gönderim fonksiyonunu inceleyelim.

```
void SPI_MasterTransmit(char cData)
{
/* İletişimi başlat */
SPDR = cData;
```

```
/* İletişimin bitmesini bekle */
while(!(SPSR & (1<<SPIF)))
;
}
```

void SPI_MasterTransmit(char cData) Bu fonksiyon cData adında char tipinde bir argüman alır. 8 bitlik değer olduğu için 8 bitlik bir değişken olması lazımdır.

SPDR = cData; SPI veri yazmacına veri yazılmasıyla iletişim başlar. Bundan önce eğer SS ayağı donanımsal değilse kendi elimizde LOW konumuna çekilmelidir.

while(!(SPSR & (1<<SPIF))) ; SPIF biti bir (1) olana kadar program sonsuz döngüye sokulur. Bu bitin bir (1) olmasıyla iletişimin bittiği anlaşılır.

Şimdi mikrodenetleyicimizi uydu aygıt olarak kullanmak için nasıl bir kod yazmamız gerekir ona bakalım. İki AVR mikrodenetleyiciyi birbirine bağlamanın en sağlıklı yollarından biri SPI protokolünü kullanmaktır.

```
void SPI_SlaveInit(void)
{
/* MISO çıkış diğerleri giriş. */
DDR_SPI = (1<<DD_MISO);
/* SPI başlat*/
SPCR = (1<<SPE);
}
```

DDR_SPI = (1<<DD_MISO); Burada sadece MISO ayağını çıkış olarak tanımladık. Bu ayak ana aygıta veri göndermek için kullanılır. Diğer tüm sinyaller ana aygıttan gelir ve burada uydu aygıtın bir denetimi yoktur.

SPCR = (1<<SPE); Sadece SPI başlat bitini kullandık. MSTR bitinin bir (1) yapılmadığına dikkat ediniz. Bu bit ile ana aygıt olarak başlatırız. Aksi halde mikrodenetleyicimiz uydu aygıt olarak tanımlanır.

```
char SPI_SlaveReceive(void)
{
/* Gelen verinin bitmesini bekle */
while(!(SPSR & (1<<SPIF)))
;
/* veri yazmacını döndür  */
return SPDR;
}
```

while(!(SPSR & (1<<SPIF))) Burada SPIF biti bir (1) olana kadar program sonsuz döngüye sokulur ve SPDR yani veri yazmacı değer olarak döndürülür.

Temel seviyede baktığımızda oldukça basit olarak görünmektedir. Bu fonksiyonları yürütmek için SPI kesmesini kullanmamız mikrodenetleyiciyi gereksiz yere meşgul etmekten kurtarır. Bu birimlerde ayrı kesmelerin olması çok faydalı bir özelliktir. Kesmelerin nasıl kullanılacağına dair bilgiyi önceki derslerimizde bulabilirsiniz. Burada SPI derslerini bitirsek de Arduino kodlarını incelediğimiz sıra Arduino'nun SPI kütüphanesini sizlere açıklayacağız.

I2C (TWI) İletişim Protokolü

50. derse başlarken önümüzdeki konuları şöyle bir gözden geçirelim, Zor konulardan geriye sadece I2C iletişim protokolü kalmış oluyor. Bu da basit bir prensipte olduğu için aslında çok da zor sayılmaz. Fakat teknik veri kitapçığıda çok tafsilatlı anlatıldığı için uzun bir konu olacaktır. Teknik veri kitapçığını incelediğimizde ise geriye şu konuların kaldığını görüyoruz,

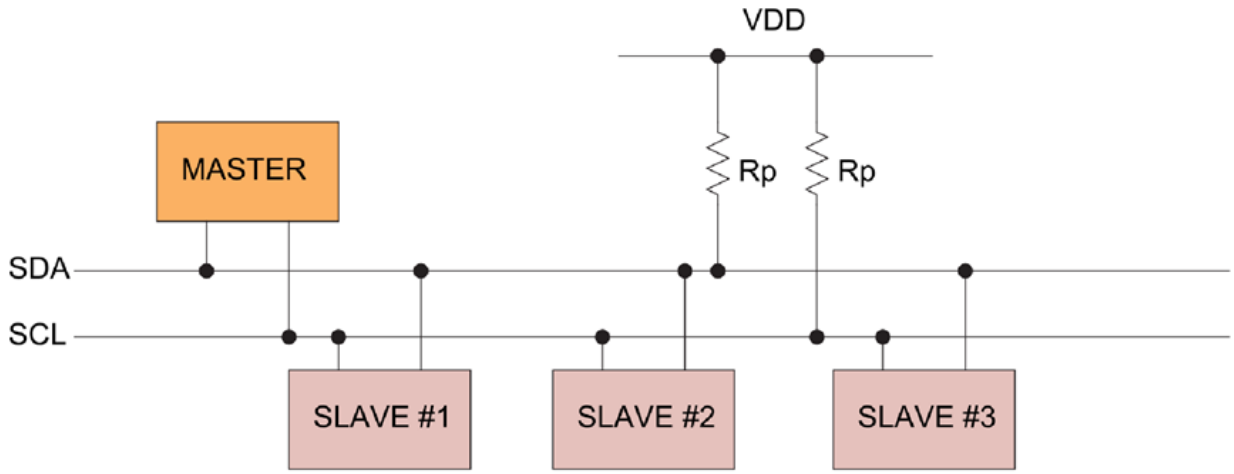
- Güç Denetimi ve Uyku Modları
- Sistem Denetimi ve Reset
- USARTSPI (Bu konu gerek görülmediği için atlanacak.)
- Debug Bağlantısı (Yine bu konu atlanacak.)
- Bootloader
- MEMPROG- Hafıza Programlama (Bu konu yazılımla halledildiği için geçilecek.)

Görüldüğü gibi AVR'ye ait konulardan geriye pek bir şey kalmamış. Geri kalan fiziksel konular ise (karakteristikler) bizi programlamada ilgilendirmediği için lazım olduğu zaman ileride bahsedilecektir.

Şimdi I2C Protokolünü anlatmaya başlayalım.

I2C iletişim protokolü lisans engelinden dolayı Atmel mikrodenetleyicilerde TWI olarak adlandırılır. Biz her ne kadar doğrusunu telaffuz etmeye çalışsak da TWI dediğimiz zaman sizin I2C olarak anlamanız gereklidir. Bu protokolü Philips firması kendi ürettiği entegre sistemler arasındaki düşük hızda iletişimi gerçekleştirmek için geliştirmiştir. Pratik bir protokol olduğu için çoğu mikrodenetleyici ve dijital denetim birimi kendi içerisinde I2C iletişim birimini bulundurur. Bu protokolde tüm veri alışverişi sadece 2 hat üzerinden sağlanır. Üstelik asenkron bir iletişim değil senkron bir iletişim söz konusudur. İki hat üzerinde çalışan protokole bağlı cihazların her birinin farklı bir adresi vardır. Bir I2C sistemine en fazla 119 aygıt bağlanabilir. I2C aygıtların 7 bitlik bir adres değeri olduğu gibi bu aygıtların belli bir adres sınırı vardır. Her aygıta kafamıza göre adres veremeyiz. Bu adres programlama işi aygıtların adres ayakları vasıtasıyla yapılır.

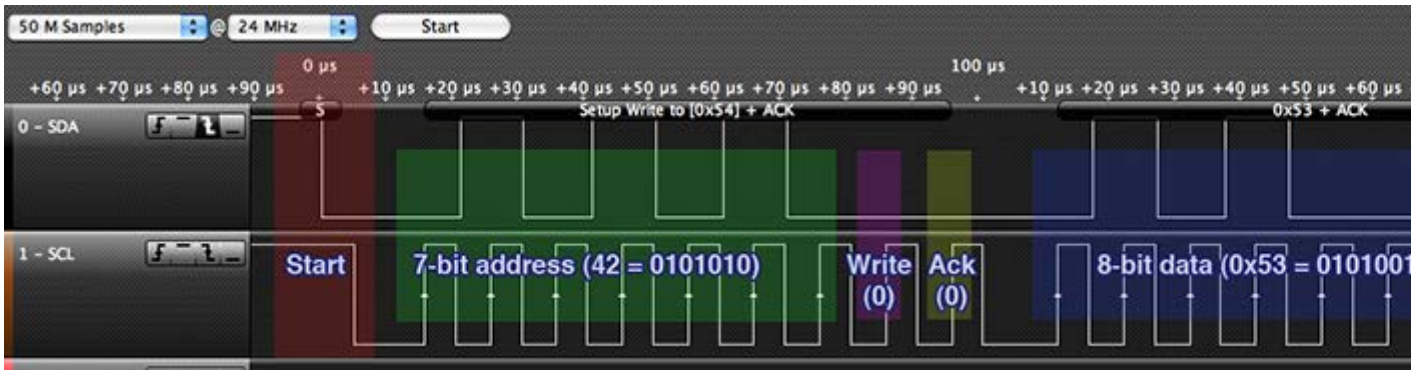
I2C Protokolünde iki adet hattın olduğunu söylemiştik. Bu hatların biri SDA yani Veri hattı öteki ise SCL yani Saat hattıdır. Bu iki hatta bütün aygıtlar bağlanır. Yine SPI Protokolünde olduğu gibi Ana ve Uydu aygıt olmak üzere iki ayrı aygıt kategorisi vardır. Ana aygıt uydu aygıtlar üzerinde denetimi sağlar. Örnek bir I2C devresi şekildeki gibidir.



Resim: http://www.analog.com/-/media/analog/en/landing-pages/technical-articles/i2c-primer-what-is-i2c-part-1-/36684.png?la=en&w=900

Burada SDA ve SCL ayaklarına Pull-up dirençlerinin bağlandığına dikkat edin. Sağlıklı bir iletişim için bu dirençleri bağlamak gereklidir. Besleme haricinde bütün bağlantılar için sadece iki hattın olduğuna dikkat ediniz.

Şimdi I2C protokolünde örnek bir veri gönderiminin lojik analizör grafiğine bakalım.



Resim: http://www.gammon.com.au/images/I2C_logic_analyzer_1.png

Bu resimde önemli noktalar yazı ile belirtilmiştir. Şimdi bu noktaları madde madde açıklayalım.

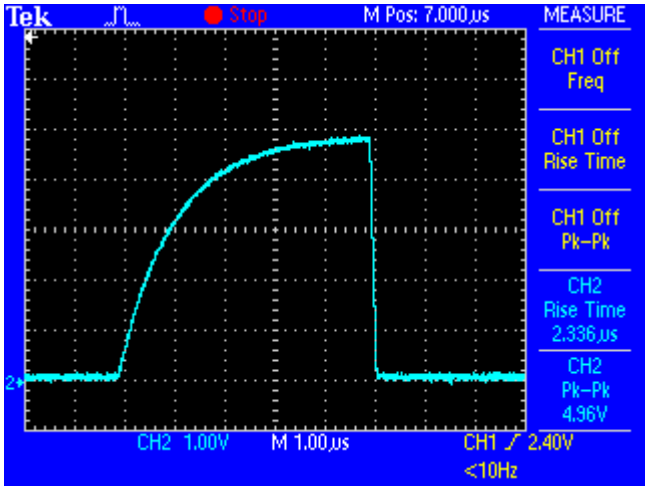
- Start noktasında SCL (Seri Saat) ayağı HIGH konumdayken SDA (Seri Veri) ayağı LOW konumuna çekilir. Böylelikle iletişim başlatılmış olur.
- Uydu aygıta veri göndermek için öncelikle aygıtın 7 bitlik adresini yazmak gereklidir. Burada baş bit öncelikli olarak gönderilir. Bu grafikte adres değeri 42'dir. Eğer herhangi bir uydu aygıt bağlı değilse veya adres gerektirmiyorsa bu duruma adres görmezden gelinir ve

SDA ayağı pull-up direnci sayesinde HIGH konmda kalır. Bu NAK (Negatif Onaylama) olarak sayıdır. Bu yazılım tarafından denetlenebilir.

- Gönderilecek bayt verisi sonrasında aynı hattan gönderilir. Yine üst bit en önce gönderilmektedir. Bu grafikte bu değer 53'dür.
- Bundan sonra diğer baytlar da gönderilebilir fakat burada gösterilmemiştir.
- Gönderim Stop durumuna gelme ile biter. SCL HIGH durumda iken SDA ayağı HIGH konuma çekilir ve iletişim bitmiş olur.

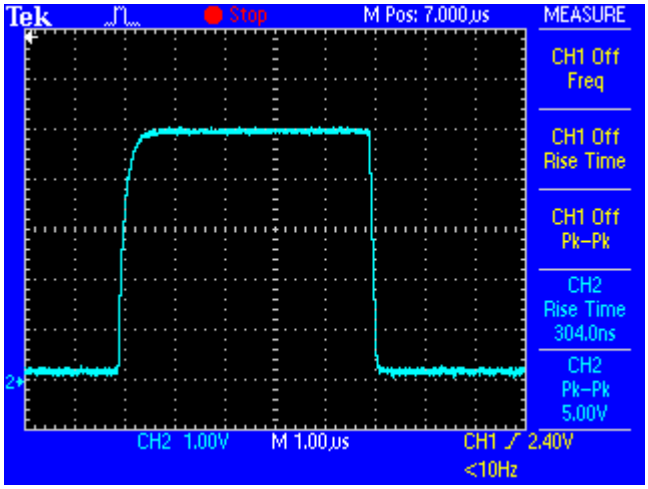
Düzgün bir kare dalga elde etmek istiyorsak 4.7K direnç ile pull-up olarak hattın beslemeye bağlanması gereklidir. 2.2K veya 1K direnç de kullanılabilir. Direncin değeri düştükçe çekilen akım artsa da dalganın düzgünlüğü de artmaktadır.

Şimdi iki adet osiloskop görüntüsü ile bu ikisi arasındaki farkı açıklayalım. İlk görüntüde sadece dahili pull-up dirençleri etkin hale getirilmiş ve herhangi bir dış pull-up direnci takılmamıştır.



Resim: http://www.gammon.com.au/images/2.2K_pullup.png

İkinci görüntüde ise 2.2K harici pull-up direnci hatta bağlanmıştır. Öncekine göre ne kadar düzgün bir kare dalga elde edildiğini göreceksiniz.



Resim: http://www.gammon.com.au/images/2.2K_pullup.png

Bir sonraki dersimizde teknik veri sayfası üzerinden I2C iletişim protokolünü anlatmaya devam edeceğiz.

TWI (I2C) Kütüphanesi

AVR derleyicisinin içerisinde bir I2C kütüphanesi bulunmasa da bir programcının yazmış olduğu ve tüm AVR aygıtlara uyumlu bir kütüphane mevcuttur. AVR I2C kütüphanesini aşağıdaki bağlantıdan indirebilirsiniz.

http://homepage.hispeed.ch/peterfleury/i2cmaster.zip

Bu kütüphaneyi programımıza dahil etmek için şu komutu kullanmamız gereklidir.

```
#include <i2cmaster.h>
```

Bu kütüphane ana aygıt (Master) kütüphanesidir ve aygıta bağlı diğer uydu aygıtlar ile iletişim kurmamızı sağlar. Çoğu uygulamamızda mikrodenetleyiciyi ana aygıt olarak kullanacağımız için bu kütüphane neredeyse çoğu işimizi görecektir.

I2C Kullanırken SDA ve SCL ayaklarını 4.7K direnç ile beslemeye bağlamayı unutmayın.

Makrolar

Bu kütüphanede iki makro bulunup i2c_start() ve i2c_rep_start() fonksiyonları ile beraber kullanılır. Hangi özellikte kullanacaksak bu makroları argümanlara “+” operatörü ile eklememiz lazımdır. Bu veri yönünü belirlemek için kolaylaştırılmış bir özellik olarak karşımıza çıkar. Makrolar ise şu şekildedir.

```
#define I2C_READ 1
#define I2C_WRITE 0
```

Şimdi kütüphane fonksiyonlarını teker teker açıklayalım.

void i2c_init (void)

Bu fonksiyon I2C ana aygıt arayüzünü başlatmaya yarar. Sadece bir kere başta çalıştırılması yeterlidir. Fonksiyonun örnek söz dizimi şu şekildedir.

```
i2c_init();
```

unsigned char i2c_start(unsigned char addr)

Bu fonksiyon belirtilen adrese bağlanmak için START durumunu başlatır. İletişimi başlatmak için öncelikle uydu aygıtın adresini bilmeli ve sonrasında ise bu adres üzerinden START durumunu başlatmamız gereklidir. Böylelikle sonrasında veri göndermemiz mümkün olur. Fonksiyon 0 ve 1 değerlerini döndürür. Eğer aygıta erişim sağlandı ise 0,

aygıta erişim sağlanamadıysa 1 değerini döndürür. Böylelikle bağlantı hataları tespit edilmiş olur. Fonksiyonun örnek söz dizimi şu şekildedir.

```
i2c_start(0x42+I2C_READ);  // OKUMA İÇİN
i2c_start(0x42+I2C_WRITE); // YAZMA İÇİN
```

unsigned char i2c_rep_start(unsigned char addr)

Bu fonksiyon yukarıdaki fonksiyon ile aynı görevi görse de farkı tekrarlayan START durumu oluşturmasıdır. Böylelikle tek bir veri değil çoklu bayt verisi gönderme imkanımız olur. Yine bu fonksiyon bir adres değerini argüman olarak alır ve başarılı olunduğunda 0, başarısız olunduğunda ise 1 değerini döndürür. Fonksiyonun örnek söz dizimi şu şekildedir.

```
i2c_rep_start(0x24+I2C_READ); // OKUMA İÇİN
i2c_rep_start(0x24+I2C_WRITE); // YAZMA İÇİN
```

void i2c_start_wait (unsigned char addr)

Bu fonksiyon START durumunu başlatır ve adres ve transfer yönünü gönderir. Eğer aygıt meşgul ise aygıt hazır olana kadar ACK durumunu sürekli kontrol eder.

```
i2c_start_wait(0x42);
```

unsigned char i2c_write (unsigned char data)

Bu fonksiyon I2c aygıtına bir bayt gönderir ve gönderim başarılı olduğunda 0 değerini döndürür. Eğer başarısız ise 1 değerini döndürür. Örnek söz dizimi şu şekildedir.

```
i2c_write(0xF0);
```

unsigned char i2c_readAck (void)

I2C aygıtından bir bayt okur ve devamı için aygıta istek yollar. Örnek söz dizimi şu şekildedir.

```
okunan_veri = i2c_readAck();
```

unsigned char i2c_readNak (void)

Bu fonksiyon bağlı aygıttan bir bayt okur ve ardından STOP durumunu başlatır. Yani bir bayt okuyarak iletişimi bitirir. Örnek söz dizimi şu şekildedir.

```
okunan_veri = i2c_readNak();
```

unsigned char i2c_read (unsigned char ack)

Bu fonksiyon yukarıda yürütülen fonksiyonların ikisini yürütebilir. ack argümanına yazdığımız 1, i2c_readAck fonksiyonunu çağırırken 0 ise i2c_readNak fonksiyonunu çağırır. 1 yazıldığında aygıttan verinin devamını talep eder, 0 yazıldığında ise tek bir okuma yapıp veri aktarımını durdurur. Fonksiyonun örnek söz dizimi şu şekildedir.

```
okunan_veri = i2c_read(1);
okunan_veri = i2c_read(0);
```

I2C kütüphanesinin bütün fonksiyonları bu kadardır. Bu kütüphane çoğu işimizi görür niteliktedir. Daha ileri işler için ise yazmaçlar üzerinde çalışıp kendi fonksiyonlarımızı yazmamız gereklidir. İleri seviye çalışmalar için AVR LibC'nin twi.h başlık dosyasında bit maskesi tanımları mevcuttur. I2C üzerinde çalışanların ihtiyacı olan bu başlık dosyasını şuradan inceleyebilirsiniz. https://www.nongnu.org/avr-libc/user-manual/group__util__twi.html

I2C konusunu burada bitirmiş olduk ve yeni konumuza geçeceğiz. Şimdiye kadar yazılan derslerden faydalanarak pek çok proje yapabilirsiniz fakat biz eksik bir konu bırakmamak istiyoruz. Seri olarak yazdığımız derslerden sonra zamanla daha ayrıntı konuları işleyeceğimiz makaleleri de yazacağız.

Güç Tasarrufu

Güç tasarrufu batarya ile beslenebilen ve taşınabilir uygulamalar için oldukça önem arz etmektedir. Sabit bir besleme kaynağına bağlı bir uygulamada çok lazım olmasa da 200-300 mili amper bir batarya veya pil ile besleme sağladığımız ve uzun süre çalışması gereken bir cihazda tasarruf edilecek birkaç mili amper bile uzun vadede oldukça önemli olacaktır. ATmega mikrodenetleyiciler kendi başlarına oldukça az bir elektrik tüketimine sahip olup en çok elektrik tüketimini ise giriş ve çıkış portları vasıtasıyla gerçekleştirir. Mikrodenetleyicinin çıkış portundan onlarca led lambayı yakıyorsak bu konuda güç tasarrufunu mikrodenetleyicide beklememiz pek doğru olmaz.

Burada anlatacağımız güç tasarrufu zaten oldukça düşük güç tüketimine sahip olan mikrodenetleyicinin ne kadar daha az elektrik harcayacağıdır. Bazı dijital sistemler o kadar az elektrik tüketir ki ufak bir pille yıllar boyu çalışabilir. Örneğin, bilgisayarlarımızdaki BIOS pili buna bir örnektir. BIOS pili aslında bir gerçek zaman saatini besleyerek bilgisayarın saat ve tarih verisini güncel tutmaya yarar. BIOS pillerinin ortalama çalışma süresi 5 yıldır.

ATmega mikrodenetleyicilerin bazı birimlerini kapatarak gereksiz güç sarfiyatını önlemiş oluruz. Bir mikrodenetleyici olarak tek başına bir işlemci değil bu işlemciye bağlı ondan fazla çevre biriminden oluştuğu için her birim bağımsız olarak çalıştığı sürece beslemeden güç almaktadır. Sadece işlemcinin tükettiği elektrik söz konusu değildir.

Güç tasarrufu sadece güç tasarrufu amacıyla kullanılmaz. Mikrodenetleyici içerisinde çalışan bu birimler elektrik gürültüsüne sebep olmaktadır. Analog devrelerde ve analog işlemlerde bu gürültü hissedilmektedir. Analog dijital çevirimin doğru olması için de "ADC Noise Reduction" adında bir güç tasarrufu modu vardır.

Güç tasarrufu hakkında ipuçlarını önceki derslerde çok az da olsa dile getirmiş olsak da bu derste en büyük güç tasarrufu yöntemi olan uyku modlarını anlatacağız. Çevre birimlerini anlatırken her birimin kontrol yazmacında bir etkinleştirme biti olduğundan bahsetmiştik. Bu birimlerin sürekli etkin olmaları yerine ayrı bir bit ile etkinleştirmeleri başta sadece ayak seçimi ile ilgili görünse de aslında bir sebebi de güç tasarrufundan dolayıdır.

AVR mikrodenetleyicilerdeki uyku modları aşağıdaki tabloda özetlenmiştir.

	Active Clock Domains					Oscillators		Wake-up Sources							Software BOD Disable
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT and PCINT	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	
Idle			Yes	Yes	Yes	Yes	Yes[2]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
ADC Noise Reduction				Yes	Yes	Yes	Yes[2]	Yes[3]	Yes	Yes[2]	Yes	Yes	Yes		
Power-down								Yes[3]	Yes				Yes		Yes
Power-save					Yes		Yes[2]	Yes[3]	Yes	Yes			Yes		Yes
Standby[1]						Yes		Yes[3]	Yes				Yes		Yes
Extended Standby					Yes[2]	Yes	Yes[2]	Yes[3]	Yes	Yes			Yes		Yes

Burada iki ayrı bölüm vardır. Bunlardan biri saat sinyali ve osilatörler diğeri ise uyandırma kaynaklarıdır. Uyku moduna göre bazı osilatörler ve saat sinyalleri devre dışı bırakılır. Uykuda çalışmayan bir mikrodenetleyicinin ise kendini uyandırması için bir dış uyarana ihtiyaç vardır. Bazen bu bir ayak okuma ile bazen de dışarıdan bağlı bir düşük frekanslı osilatörle olabilir. Bu uyku modlarını yine yazmaçlar vasıtasıyla denetlememiz gereklidir.

Bu altı uyku modundan birine girmek için uyku modu denetim yazmacındaki uyku modu etkinleştirme bitini bir (1) yapmamız gereklidir. Uyku modunu seçmek için ise yine uyku modu seçme bitleri üzerinde işlem yapmamız gereklidir.

Uyku modunun özelliklerinden biri ise kesmelerin mikrodenetleyiciyi uyandırabilmesidir. Böyellikle uyandırma kaynaklarından biri de kesmeler olmuş olur. Mikrodenetleyici kesme yürütüldükten sonra kendini otomatik olarak uyutur.

Bir sonraki derste güç tasarrufu ve uyku modlarını ayrıntılı olarak anlatmaya devam edeceğiz.

Uyku Modları ile Güç Tasarrufu

AVR denetleyicilerde uyku modlarından önceki yazımızda bahsetmiştik. Bu modların geniş açıklamasını ise bu yazıda yapacağız. Dersin devamında ise yazmaçları açıklayıp konumuzu bitireceğiz. Bu konu kolay konulardan biri olduğu için anlamanız için pek fazla çaba sarf etmeyeceğim. Derslerin başından beri yaptığımız gibi yazmaçların belli bitlerini değiştirerek istediğimiz işi yaptırabiliyoruz. Sizin bu modların nasıl olduğu ve nasıl çalıştığı konusunda bilgi sahibi olmanız için biraz daha ayrıntılı olarak modları anlatmaya devam edelim.

BOD Devre Dışı Bırakma

Mikrodenetleyicide “Brown-out Detector” adı verilen kararma algılayıcısı bulunmaktaydı. Bu algılayıcı mikrodenetleyiciye giden beslemede düşüş olması durumunda (örneğin 2.7V aşağısı) mikrodenetleyiciyi otomatik olarak resetliyordu. Bu mikrodenetleyicinin yetersiz beslemeye bağlı kararsız çalışmasını önlemek içindi. Kararma algılayıcısı sigorta bitleri tarafından denetlendiği için bunu programlarken programlayıcı arayüzünden seçip

yapabiliriz. Programlama yazılımı olarak AVRDUDESS programını tavsiye ederim. Arayüz olarak kullanışlı olan bu programda gerekli sigorta ayarlarını yapabilirsiniz.

Sigorta hesaplamaları ise şu bağlantıdan yapılabilir. Sigorta değerleri hesaplandıktan sonra program vasıtasıyla yazdığımız programdan ayrı olarak mikrodenetleyiciye yüklenir. Sigorta bitlerini daha sonra anlatacağımız için burada bırakalım.

http://www.engbedded.com/fusecalc

Sigorta ile etkin hale getirilen kararma algılayıcısı sürekli besleme gerilimini ölçerek belli bir güç tükettiğinden bunu sonraca yazılım ile kapatmamız mümkündür. MCUCR yazmacının 6. biti olan BODS biti bir (1) yapılarak kararma algılayıcısı devre dışı bırakılabilir. Bu devre dışı bırakma ile uyku modunda fazladan güç tasarrufu sağlanır.

Bekleme Modu (Idle)

SM bitleri “000” yapıldıında uyku modu olarak bekleme modu seçilir. Bekleme modunda mikroişlemci durdurulur fakat SPI, USART, Analog Karşılaştırıcı, WDT ve kesmeler çalışmaya devam eder. Bu uyku modu işlemci ve flash hafıza saatini durdurur fakat diğer saatlerin işlemesine izin verir. Bekleme modunda denetleyici uyandırıcılardan biri olan kesmeler tarafından rahatlıkla uyandırılabilir.

ADC Gürültü Önleme Modu

Arduino’da ADC gürültüsünü önlemek için meşhur bir program vardı. Bu program hali hazırda gürültülü olan ADC değerlerini toplayıp bu gürültülü değerlerin ortalamasını alarak gürültüyü önlediğini iddia ediyordu. Aslında bu bir yönden işe yarar olsa da mevcut gürültü hiçbir zaman ortadan kalkmış olmuyordu. Üreticiler bu gürültünün farkında olup programcıları böyle yollara itmemek için gürültü önlemeye yönelik özel bir mod getirmiştir. SM bitleri “001” yapıldığında denetleyici ADC gürültü önleme moduna girer. İşlemci durdurulsa da ADC, harici kesmeler, TWI, TC2 ve WDT işlemeye devam eder. Böylece diğer birimlerin ve işlem biriminin yaptığı gürültü ölçüme etki etmemiş olur. Hassas analog işlemlerde bu gürültü bazen hiç doğru çalıştırmayacak derede etkiye sahip olduğu için bu özellik oldukça faydalı olacaktır.

ADC gürültü önleme modundan ADC çevirim tamamlandı kesmesi ile çıkabilir ve okunan değeri kaydedebiliriz. Bundan başka harici kesme, WDT sistem reset, WDT kesmesi, kararma reseti, TC2 kesmesi, TWI adres eşleşmesi, EEPROM hazır kesmesi gibi durumlarda da uyku modundan uyanılabilir.

Power-Down (Güç Kesme) Modu

SM bitleri “010” yapıldığında uyku modu olarak Power-Down modu seçilmiş olur. Bu modda harici osilatör durur ve dış kesmeler, TWI adres izlemesi , WDT (Watch-Dog Timer) çalışmaya devam eder. Ancak aşağıdaki durumlardan biri mikrodenetleyiciyi uyandırabilir.

- Dış reset
- WDT Sistem Reseti
- WDT kesmesi
- Kararma reseti
- TWI adres eşleşmesi
- INT ayağından dış kesme
- Ayak değişme kesmesi
  Bu uyku modu temel olarak bütün üretilen saat sinyallerini durdurur. Sadece asenkron çalışan modüllerin çalışmasına izin verir.

  Bir sonraki yazıda bu modları anlatmaya devam edeceğiz.

Güç Tüketimini En Az Seviyeye İndirmek

Öncelikle önceki derste yarım bıraktığımız uyku modlarını anlatarak derse başlayalım.

Güç Tasarrufu (Power-Save) Modu

SM bitleri “011” yapıldığında denetleyici Power-Save modunda uykuya geçer. Bu mod power-down moduyla aynı olsa da bir yönden farklılığı vardır. Eğer TC2 zamanlayıcısı etkinleştirildi ise bu zamanlayıcı uyku süresinde çalışmaya devam eder. Taşma ve karşılaştırma eşleşmesi kesmelerinde denetleyici uyandırılır. Böylelikle mikrodenetleyici belli aralıklarla uyanıp kod işler ve sonrasında tekrar kendini derin uykuya sokar. TC2 zamanlayıcısına dışarıdan bir sinyal vermekle bu aralık uzatılabilir. Böylelikle güç tasarrufu daha da artırılmış olur. Eğer TC2 zamanlayıcısı işletilmiyorsa power-down modunun kullanılması tavsiye edilir.

Bekleme (Standby) Modu

SM bitleri “110” yapıldığı zaman denetleyici bekleme moduna girer. Bu mod power-down moduyla aynı olsa da osilatör çalışmaya devam eder. Aygıt altı saat sinyalinde uyanır.

Genişletilmiş Bekleme (Extended Standby) Modu

SM bitleri “111” yapıldığında denetleyici genişletilmiş bekleme moduna girer. Bu mod power-save modu ile aynı olsa da bu modda osilatörler çalışmaya devam eder.

Güç Tasarrufu Yazmacı

Bu yazmaç çeşitli çevre birimlerini durdurarak güç tüketimini düşürmeyi sağlar. PRR (Power Reduction Register) yazmacındaki biti tekrar sıfır (0) yaparak o çevre birimini kullanmaya devam edebiliriz.

Güç Tüketimini Asgari Seviyeye Çekmek

Güç tüketimini en dibe çekmek için çeşitli yollar mevcuttur. Uyku modları mümkün olduğu kadar fazla kullanılmalıdır. Çünkü uyku modları en büyük güç tasarrufunu sağlayan yollardan biridir. Ayrıca uyku modunda bizim çalıştıracağımız denetleyici özelliklerinin de mümkün olduğu kadar az olması gereklidir. Uyku modunda mikrodenetleyiciye neredeyse hiçbir şey yaptırmamak lazımdır.

Eğer ADC etkinleştirilirse ADC birimi tüm uyku modlarında çalışmaya devam eder. Güç tasarrufu için uyku moduna girmeden önce ADC'nin kapatılması gereklidir.

Bekleme (Idle) moduna giriliyorsa eğer kullanılmayacaksa analog karşılaştırıcı kapatılmalıdır. ADC gürültü önleme moduna girilirken ise analog karşılaştırıcı muhakkak kapatılmalıdır. Diğer modlarda analog karşılaştırıcı otomatik olarak kapatılır. Bu birimlerin nasıl açılıp kapatılacağını yazmaçlarından bahsettiğimiz derslerde açıkladık.

Kararma algılayıcısı uygulamada gereksiz ise kapatılması gereklidir. Eğer sigortadan etkinleştirilmiş ise tüm uyku modlarında çalışacak ve güç tüketecektir. Derin uyku modlarında ciddi bir oranda güç tüketimi olarak karşımıza çıkacaktır. Bunu devre dışı bırakmayı önceki derste açıklamıştık.

WDT zamanlayıcısı gerek duyulmuyorsa kapatılması gereklidir. Eğer WDT zamanlayıcısı açık ise tüm uyku modlarında çalışacaktır. Bu zamanlayıcısı sistem denetim ve reset konusunda ileride açıklayacağız.

Bütün port ayakları uyku moduna girilmeden önce asgari güç tüketimine göre ayarlanmış olmalıdır. Dijital giriş tamponunu devre dışı bırakmak analog işlemlerde güç tasarrufu sağlayacaktır.

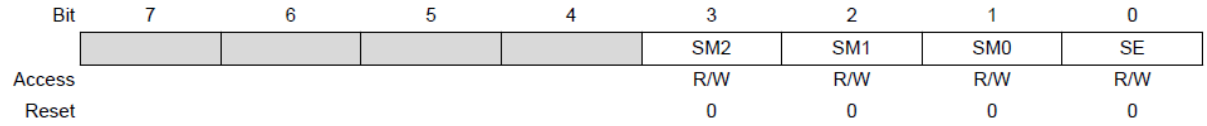
Eğer projemiz hız gerektirmiyorsa harici kristal osilatör yerine dahili osilatörü kullanarak da güç tüketimini azaltabiliriz. Hız düştükçe güç tüketimi her alanda düşecektir.

Güç tasarrufuna dair anlatacağımız bilgiler bu kadardı. Geriye ise bu konuda kullanılacak yazmaçlar kalmıştır. Yazmaçları ise sonraki dersimizde anlatacağız.

Güç Tasarrufu Yazmaçları ve Kütüphanesi

AVR mikrodenetleyicileri sadece yazmaç denetimi ile uykuya sokamayız. Üreticiler uyku moduna geçmek için özel bir mikroişlemci komutu belirlemiştir. Bu mikroişlemci komutunu C dilinde kullanmak için avr/sleep.h başlık dosyasını kullanırız. Bütün bu ayar ve etkinleştirmeler sonunda SLEEP komutu ile mikrodenetleyici uykuya geçer. Dersin sonunda bundan bahsedeceğimiz için şimdi yazmaçları anlatarak konumuza devam edelim.

SMCR – Uyku Modu Denetim Yazmacı



Bit	7	6	5	4	3	2	1	0
					SM2	SM1	SM0	SE
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 3, 2, 1 – Uyku Modu Seçme Biti

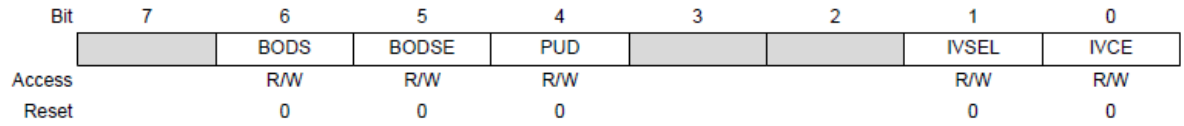
Bu bitler aşağıdaki tabloya göre uyku modunu seçmemizi sağlar.

SM2, SM1, SM0	Uyku Modu
000	Idle
001	ADC Gürültü Önleme
010	Power-Down
011	Power-Save
100	Rezerve
101	Rezerve
110	Standby
111	Extended Standby

Bit 0 – SE : Uyku Etkin

Bu bit mikrodenetleyicinin uyku moduna girmesi için gerekli izni verir. Uyku moduna bu biti bir (1) yaparak girmeyiz. Uyku modu için özel mikroişlemci komutu vardır. Fakat bu komutun işletilmesi için bu bitin bir (1) olması lazımdır. SLEEP komutu işletilmeden hemen önce bu biti bir (1) yapmak daha iyi olacaktır.

MCUCR – Mikrodenetleyici Denetim Yazmacı



Bit	7	6	5	4	3	2	1	0
		BODS	BODSE	PUD			IVSEL	IVCE
Access		R/W	R/W	R/W			R/W	R/W
Reset		0	0	0			0	0

Bit 6 – BODS – BOD (Brown-out Detection) Uyku

Bu bit bir (1) yapıldığında kararma algılayıcısı uyku sırasında devre dışı bırakılır. BODSE biti ise bir etkinleştirme bitidir ve güvenlik amacıyla öncelikle iki bit aynı anda bir (1) yapılmalı ve dört saat çevrimi içerisinde BODSE biti sıfır (0) yapılmalıdır. BODS biti üç saat

çevrimi boyunca etkin halde olur. Bu durumda ise uyku komutu işletilmelidir. BODS biti üç saat çevrimi sonunda otomatik olarak sıfırlanır.

Bit 5 – BODSE : BOD Uyku Etkin

Bu bitin görevini yukarıda açıkladık.

PRR – Güç Tasarrufu Yazmacı

Bit	7	6	5	4	3	2	1	0
	PRTWI0	PRTIM2	PRTIM0		PRTIM1	PRSPI0	PRUSART0	PRADC
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Bit 7 – PRTWI0 : TWI Güç Tasarrufu

Bu biti bir (1) yaparsak TWI modülü çalışmayı durdurur. Giden saat sinyalini kesmekle bu gerçekleşir. Tekrar etkinleştirmek için modülü yeniden başlatmak gerekir.

Bit 6 – PRTIM2 : TC2 Güç Tasarrufu

Bu biti bir (1) yaparsak TC2 zamanlayıcısı senkron modda çalışmayı durdurur. Zamanlayıcı etkinleştirildiğinde tekrar çalışmaya devam eder.

Bit 5 – PRTIM0 : TC0 Güç Tasarrufu

Bu biti bir (1) yaparsak TC0 zamanlayıcısı senkron modda çalışmayı durdurur. Zamanlayıcı etkinleştirildiğinde tekrar çalışmaya devam eder.

Bit 3 – PRTIM1 : TC1 Güç Tasarrufu

Bu biti bir (1) yaparsak TC1 zamanlayıcısı senkron modda çalışmayı durdurur. Zamanlayıcı etkinleştirildiğinde tekrar çalışmaya devam eder.

Bit 2 – PRSPI0 : SPI Güç Tasarrufu

Bu biti bir (1) yaparsak SPI modülü çalışmayı durdurur. Tekrar etkinleştirmek için modülü yeniden başlatmak gereklidir.

Bit 1 – PRUSART0 : USART Güç Tasarrufu

Bu biti bir (1) yaparsak USART çalışmayı durdurur. Tekrar etkinleştirmek için modülü yeniden başlatmak gereklidir.

Bit 0 – PRADC : ADC Güç Tasarrufu

Bu biti bir (1) yaparsak ADC modülü kapatılır. Kapatılmadan önce ADC devre dışı bırakılmalıdır. (ADEN biti)

Yazmaçlar bitmiş oldu. Şimdi SLEEP komutunu nasıl işleteceğimiz hakkında kısa bir bilgi verelim.

Uyku işlemlerini yapmak istiyorsak öncelikle programımıza şu başlık dosyasını dahil etmemiz gereklidir. sleep.h başlık dosyası derleyicinin içinde gelmektedir.

```
#include <avr/sleep.h>
```

Şimdi fonksiyonları kısaca açıklayalım.

```
sleep_bod_disable();
```

Kararma algılayıcısını devre dışı bırakır. (Her aygıtta geçerli değildir.)

```
sleep_cpu();
```

İşlemciyi uyku moduna sokar. Assembly dilindeki sleep komutu ile aynıdır. SE biti önceden bir yapılmalıdır. Uykunun ardından tekrar sıfır yapılması önerilir.

```
sleep_disable();
```

SE bitini sıfırlar. Böylelikle denetleyici bir daha uykuya bu bit bir (1) yapılana kadar girmez.

```
sleep_enable();
```

SE bitini bir (1) yapar.

```
sleep_mode();
```

SE bitini önceden bir (1) yapar ve ardından aygıtı uykuya sokar.

Yazmaçları öğrendiğimiz için sadece SLEEP mikroişlemci komutunu kullanmamız yeterlidir. Bunun için ise yazmaç ayarlarını yaptıktan sonra sleep_cpu() fonksiyonunu kullanacağız.

SLEEP komutu ise aşağıdaki bitlerden ibarettir. Bu bitler program hafızasına yazılır ve mikroişlemcinin anlayacağı makine dilindedir.

```
1001 0101 1000 1000
```

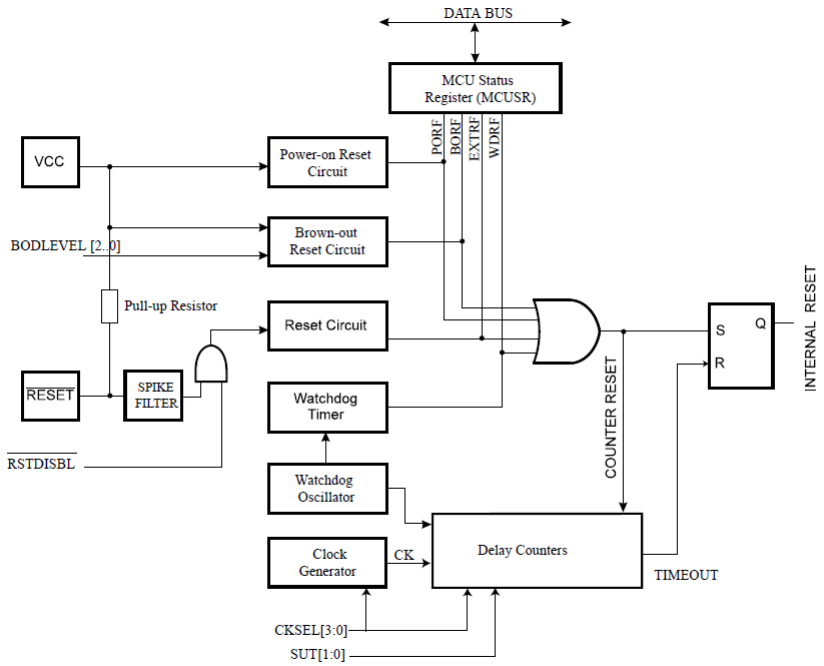
Güç tasarrufu ve uyku konusunu bitirmiş olduk. Şimdi sırada tek bir konumuz kaldı sayılır. Sistem denetimi ve yeniden başlatma konusunun ardından derslerimizi bitirmiş olacağız. Geriye kalan konuları ise derslerden bağımsız olarak daha sonra ele alacağız.

Yeniden başlatma (Reset) bilgisayarlardan tanıdık olduğumuz üzere kararsızlaşan bir sistemi tekrar kararlı hale sokmak için yürütülen bir düzenleme eylemidir. Yeniden başlatma özelliği olmayan makinalarda bile bunu el ile yapmamız mümkün olmaktadır. Kapatıp açmak, çoğu elektronik aleti onarmak için başvurduğumuz ilk yöntemlerden biridir.

Yeniden başlatma sürecinde giriş ve çıkış yazmaçları sıfırlanır, program ise reset vektöründen itibaren çalışmaya başlar. Bilgisayarlarda yeniden başlatma işletim sisteminden veya reset düğmesinden gerçekleştirildiği gibi AVR mikrodenetleyicilerde de çeşitli yeniden başlatma kaynakları vardır. Bu kaynaklar vasıtasıyla çeşitli yollardan yeniden başlatma işlemini gerçekleştirmiş oluruz.

Aşağıdaki listede yeniden başlatma kaynakları açıklanmıştır.

- Açılış Reseti (Power-on Reset) : Mikrodenetleyiciye çalışması için güç verildiği zaman ve bu güç de açılış reseti eşiğini geçtiği zaman yeniden başlatma işi gerçekleştirilir.
- Harici Reset : RESET ayağı belli bir süre sıfır (0) yapıldığında mikrodenetleyici kendini yeniden başlatır.
- WDT Sistem Reseti : Bekçi zamanlayıcısı periyodu geçtiği zaman gerçekleştirilen yeniden başlatmadır.
- Kararma (Brown-out) Reseti: Besleme gerilimi kararma eşiğinin altına düştüğünde gerçekleştirilen yeniden başlatmadır.
  Aşağıdaki şemada reset mantık devresinin blok diyagramını görüyoruz.



MCUSR – Mikrodenetleyici Durum Yazmacı

Bit	7	6	5	4	3	2	1	0
					WDRF	BORF	EXTRF	PORF
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 3 – WDRF – Bekçi Zamanlayıcısı Sistem Reset Bayrak Biti

Bekçi zamanlayıcısı sistem yeniden başlatma gerçekleştiğinde bu bit bir (1) konumuna gelir. Açılış yeniden başlatımında veya sıfır (0) yazılmasında bu bit eski haline gelir.

Bit 2 – BORF : Kararma Reset Bayrak Biti

Bu bit Brown-out reseti gerçekleştiğinde bir (1) konumuna gelir. Açılış reseti ile veya sıfır (0) yazma ile eski haline alır.

Bit 1 – EXTRF : Harici Reset Bayrağı

Harici Reset durumunda bu bit bir (1) konumuna gelir. Açılış reseti ile veya sıfır (0) yazma ile eski haline alır.

Bit 0 – PORF – Power-on Reset Bayrağı

Açılış resetidir. Sıfır (0) yazma ile eski haline alır.

Reset konusu bayağı kısa olduğu için bu kadarını anlatabildik. Bir sonraki derste bekçi zamanlayıcısını (watch-dog timer) ele alacağız.

Bekçi Zamanlayıcısı (Watchdog Timer)

İsmi oldukça ilginç olan bir zamanlayıcı ile karşı karşıyayız. Watchdog timer bir terim olsa da aslında pek çok yerde karşılaşacağımız gibi anlamsız ve uydurulmuş bir tabirdir. Türkçe'ye tam olarak çevrimi "Bekçi köpeği zamanlayıcısı" olup bizim kafamızda soru işaretleri bırakmaktadır. Bu İngilizce'nin zengin kelime üretme kabiliyetinden dolayı değildir. İngilizce yeni kelime üretmeye oldukça kapalı bir dil olduğundan yeni bir kelime ihtiyacı duyulduğunda böyle anlam kaymasına uğrayan uydurma kelimeler üretilir. Aynı erkek arı anlamına gelen "drone" kelimesinin uçan bir cihaz için kullanılmasında olduğu gibi böyle tabirlerin Türkçeleştirilme zorluğu anlamsızlığından dolayı olmaktadır.

Bu zamanlayıcıyı tanımlamak için böyle bir tabir kullanılmasından dolayı kısa bir dilbilim dersi vermek durumunda kalmış olduk. Şimdi zamanlayıcıyı anlatalım.

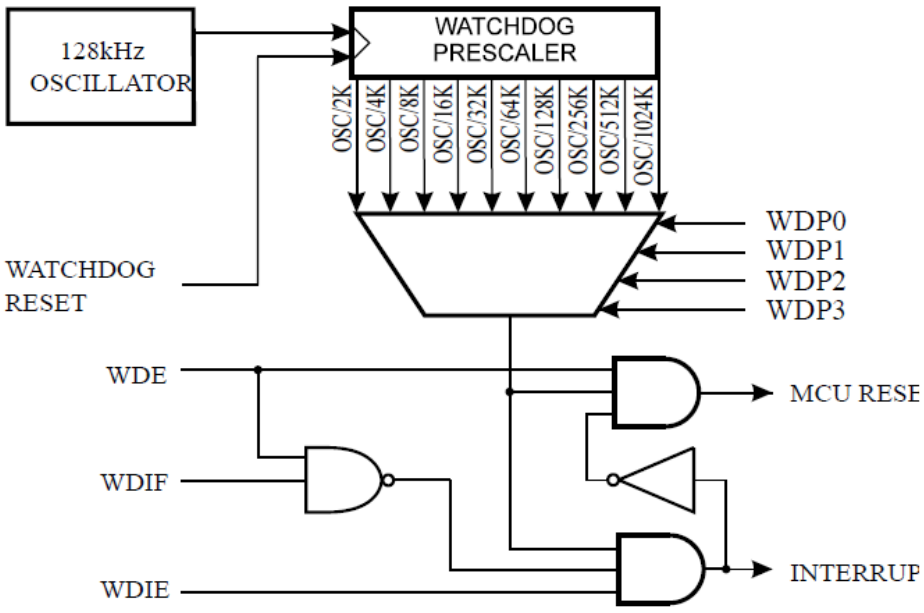
Bekçi zamanlayıcısı oldukça basit bir işleve sahip olan zamanlayıcı ünitesidir. Zamanlayıcılar eşleşme veya taşma anında bir çıkış veriyordu. Bu çıkışı reset ayağına bağlayıp mikrodenetleyiciyi her çıkış anında yeniden başlatırsak bekçi zamanlayıcısının bir kopyasını yapmış oluruz. Bekçi zamanlayıcı taşma veya eşleşme anında mikrodenetleyiciyi yeniden başlatan bir zamanlayıcıdır. Yine osilatör, ön derecelendirici, karşılaştırma yazmacı, sayaç yazmacı gibi standart zamanlayıcı birimlerine sahiptir.

Mikrodenetleyiciyi sürekli resetleyen bir sayıcıyı düzgün kullanamadıkça faydadan çok zarar göreceğimiz ortadadır. Bu düzgün kullanım ise yazılımla bu sayıcıyı sürekli sıfırlayıp resetlemesinin önüne geçmekle mümkün olur. Program düzgün çalıştığı sürece bu sayıcıyı sıfırlayarak “Mavi ekran” vermesinin önüne geçer. Eğer program sonsuz döngüye girer veya bir yerde takılıp cevap veremez konuma gelir ise bu sayıcı sıfırlanmaz ve taşarak “Mavi ekran” verip mikrodenetleyiciyi yeniden başlatır.

İşte programımız istenildiği gibi çalışmazsa bizi kurtaracak birim bekçi zamanlayıcısıdır. Program sonsuz döngüye girdiğinde veya bir yerde donduğunda bizi kurtaracak bir özellik yoktur. Ya kullanıcı tarafından el ile yeniden başlatılmalı ya da bu yeniden başlatma otomatik yolla yapılmalıdır.

Bu zamanlayıcı ayrı bir dahili osilatör ile beslenir. Kesme, sistem reset ve kesme ve sistem resetinin beraber yürütüldüğü üç ayrı çalışma modu vardır. 16 mili saniyeden 8 saniyeye kadar zaman aşımı modu vardır. Yani en fazla 8 saniye içinde sıfırlanmadığı takdirde sistemi yeniden başlatır. Donanımsal olarak sigorta bitine sahip olup sürekli açık hale getirilebilir.

Her zaman olduğu gibi öncelikle bekçi zamanlayıcısının blok diyagramını paylaşalım ve bunun üzerinden anlatmaya devam edelim.



Görüldüğü gibi zamanlayıcının kendine ait bir osilatörü bulunmaktadır. Bu osilatör 128kHz’de çalışmaktadır. Bu osilatörün saat sinyali bekçi zamanlayıcısı ön derecelendiricisine gitmektedir. Ön derecelendirici 2 ile 1024 arasında sinyali bölme işlemine tabi tutup sayaç değerini artırmaktadır. Sayacı sıfırlamak için Watchdog Reset sinyali kullanılır. Sayaç çıkışı ise kesme ve mikrodenetleyici yeniden başlatmaya bağlanmıştır.

Bekçi zamanlayıcısı hem kesme yürütebilir hem de sistemi resetleyebilir. Bunun için sayaç değerinin taşma değerine ulaşması lazımdır. Normal bir sistem çalışmasında taşma

değerine ulaşmadan önce bekçi zamanlayıcısı sıfırlanmak zorundadır. Eğer sistem sayacı sıfırlamazsa kesme ya da sistem reseti yürütülür.

Kesme modunda sayaç taştığı (zaman aşımı olduğu) zaman kesme yürütülür. Bu kesme aygıtı uyku modundan çıkarmak için kullanılabilir. Sistem reset modunda ise aygıt taşma gerçekleştiğinde yeniden başlatılır. Üçüncü mod olan kesme ve sistem reset modunda önce kesme yürütülür ve sonrasında ise sistem yeniden başlatılır. Bu modla kritik bilgiler sistem yeniden başlatılmadan önce kaydedilme şansı bulur.

Eğer WDTON sigortası programlanırsa bekçi zamanlayıcısı sürekli sistem reset modunda çalışmaya zorlanır. Güvenlik açısından bekçi zamanlayıcısı şu prodesürlerle kullanılmalıdır.

Aynı fonksiyonda bekçi zamanlayıcısı değişim etkinleştirme biti (WDCE) ve bekçi zamanlayıcısı sistem reset etkinleştirme biti (WDE) bir (1) yapılmalıdır. WDE biti öncelikle bir (1) yapılması lazımdır.

Sonraki dört saat çevrimi içerisinde WDE ve ve ön derecelendici bitleri istenildiği gibi düzenlenebilir. Fakat WDCE bitinin de sıfır (0) yapılması lazımdır. Bu aynı komut içinde yapılmalıdır.

Zamanlayıcıyı etkinleştirmek için tek bir biti açmak (WDCE) yeterli olsa da taşmadan önce kapatmak için biraz fazlaca kod yazmamış gereklidir. Zamanlayıcıyı kapatmak için şöyle bir fonksiyon kullanılmalıdır.

```
void WDT_off(void)
{
__disable_interrupt();
__watchdog_reset();
/* WDRF bitini sıfırla */
MCUSR &= ~(1<<WDRF);
/* WDCE ve WDE bitine aynı anda bir yaz. */
/* Eski ön derecelendirici ayarını muhafaza etmek daha sağlıklı olur.*/
WDTCSR |= (1<<WDCE) | (1<<WDE);
/* WDT kapat */
WDTCSR = 0x00;
__enable_interrupt();
}
```

Zamanlayıcıyı sıfırlamak için ise öncelikle avr/wdt.h başlık dosyası eklenmelidir. Aşağıdaki komutu programın başına eklememi lazımdır.

```
#include <avr/wdt.h>
```

Sonrasında ise şu fonksiyonu kullanarak zamanlayıcıyı kolaylıkla sıfırlayabiliriz.

```
wdt_reset();
```

Ön derecelendiriciyi değiştirmek için ise şöyle bir fonksiyon kullanılmalıdır.

```
void WDT_Prescaler_Change(void)
{
__disable_interrupt();
__watchdog_reset();
```

```
/* Zamanlamayı başlat (4 saat çevrimi) */
WDTCSR |= (1<<WDCE) | (1<<WDE);
/* Yeni ön derecelendirici değerini belirle 64K çevirim (~0.5 s) */
WDTCSR = (1<<WDE) | (1<<WDP2) | (1<<WDP0);
__enable_interrupt();
}
```

Bekçi zamanlayıcısını etkinleştirmek için ise şöyle bir komut kullanılmalıdır.

```
wdt_enable();
```

Yine devre dışı bırakmak için kütüphane fonksiyonu kullanmak istiyorsak şöyle bir komut kullanabiliriz.

```
wdt_disable();
```

WDTCSR – Bekçi Zamanlayıcısı Denetim Yazmacı

Bit	7	6	5	4	3	2	1	0
	WDIF	WDIE	WDP[3]	WDCE	WDE	WDP[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – WDIF : Bekçi Zamanlayıcısı Kesme Bayrak Biti

Bu bit zamanlayıcı kesme modunda çalışırken taşma olduğu sırada bir (1) konumuna geçer. Bu bit kesme fonksiyonu yürütüldüğü zaman otomatik sıfırlanır. Alternatif olarak el ile bir (1) yazarak bu biti sıfırlayabiliriz .

Bit 6 – WDIE – Bekçi Zamanlayıcısı Kesme Etkin

Bu biri bit (1) yapmak kesmeyi etkinleştirir. Aşağıda zamanlayıcının ayarlarına dair bir tablo vardır.

WDTON Sigortası	WDE Biti	WDIE Biti	Mod	Taşma Sırasında Yürütülen İşlem
1	0	0	Durdurulur	Yok
1	0	1	Kesme Modu	Kesme
1	1	0	Sistem Reset Modu	Reset
1	1	1	Kesme ve Reset Modu	Kesme ve sonrasında Reset
0	x	x	Sistem Reset Modu	Reset

Bit 5 – WDP [3] – Bekçi Zamanlayıcısı Ön derecelendiricisi

Bit 4 – WDCE : Bekçi Zamanlayıcısı Değiştirme Etkin

Bu bit zamanlanmış ayar süreci için kullanılır. WDT ayarı ve ön derecelendirici ayarını yapmak için verilen süreci bu bit ile yaparız. Bu bir çeşit güvenlik bitidir. WDE ve ön derecelendirici bitlerini değiştirmek istiyorsak bu biti bir (1) yapmak zorundayız. Bir yapıldıktan sonra donanım bu biti dört saat sinyali içerisinde otomatik sıfırlar. O yüzden hemen gereken kodu yazmamız gereklidir.

Bit 3 – WDE : Bekçi Zamanlayıcısı Sistem Reset Etkin

Eğer MCUSR içindeki WDRF biti bir (1) ise bu bit bir (1) olmaya zorlanır. WDE'yi sıfırlamak için önce WDRF'yi sıfırlamak lazımdır. Bu özellik çoklu reset yürütülmesini önlemeye yarar.

Bit 2:0 – Bekçi zamanlayıcısı ön derecelendirici bitleri

Ön derecelendirici aşağıdaki tabloya göre çalışmaktadır.

WDP[3]	WDP[2]	WDP[1]	WDP[0]	WDT Osilatör Çevirim Sayısı	Süre
0	0	0	0	2K	16ms
0	0	0	1	4K	32ms
0	0	1	0	8K	64ms
0	0	1	1	16K	0.125ms
0	1	0	0	32K	0.25s
0	1	0	1	64K	0.5s
0	1	1	0	128K	1s
0	1	1	1	256K	2s
1	0	0	0	512K	4s
1	0	0	1	1024K	8s

Böylelikle AVR konularımızı bitirmiş olduk. Artık tüm dersler bittiğine göre ileri seviye AVR programlamayı öğrenmek veya AVR programlamaya üst seviyeden başlamak isteyen programcıların faydalanabileceği eksiksiz bir ders dizisi ortaya çıkmış oldu. İlerleyen zamanlarda bu derslere yardımcı kaynak olacak şekilde kaynak makale yazmaya devam edeceğiz