Serial memory entegrelerinin kütüphanesiz kullanımı.

SPI haberleşme, 4 kablo ile 20mhz hızına kadar haberleşme yapabileceğiniz bir seri haberleşme protokolüdür.   
**ilk resim**

**İkinci resim**

**Yukarıdaki pim adlarını açıklayalım**.   
**MISO (Master In Slave Out)** – Slave den Master’a bilgi gönderilir.   
**MOSI (Master Out Slave In)** – Master dan slave’e doğru bilgi akışı.  
**SCK (Serial Clock) –** Saat frekansı her iki yönde bilgi akışına kılavuzluk eder. Model ar CLK piminin çalışmasına göre şekillenir.  
**SS (Slave Select) –** Bir veya birden fazla alıcı olması durumunda her alıcıyı seçmek için kullanılan pim.

Direkt iletişim olduğundan gelen bilgilerin doğruluğundan emin olamayabilirsiniz. Kısa mesafe haberleşmesi olduğundan CHECKSUM (doğrulama) gibi eklentileri yoktur. Bu da doğrulama yapamadığımız anlamına gelir. Sizler 2 arduino arasında bunu kullanmak isterseniz ve doğrulama ihtiyacı hissederseniz checksum sistemi gibi bir kod formatı yaratabilirsiniz. Ya da gelen datayı doğrulama olarak karşıya gönderebilirsiniz. Tabii ki bu da haberleşme hızınızı bir miktar düşürecektir.

(Checksum dediğimiz, seri bilginin sonuna eklenen tüm dizinin durumuna uygun bir toplam işlemidir.   
Mesela, gönderilen karakterlerin çift olanlarının toplamı diyelim. BABA kelimesini göndermek istersek. ASCII tablosundan B=0x66 A=0x65 tir. Yani göndereceğimiz bilgi   
BABA = 0x66,0x65,0x66,0x65 burada 66 çift rakam olduğundan ve 2 tane olduğundan, diğer arduinoya göndereceğimiz bilgi   
BABA = 0x66,0x65,0x66,0x65,0x02 sayısı olursa gelen bilgi ile son verinin karşılaştırmasını alıcı arduino hesaplar ve doğruladığında karşıya bir doğrulandı bilgisi gönderebilir. Karmaşık checksum bilgisine gerek kalmayabilir. )

* SPI kütüphanesinde bazı kabuller vardır. İstediğiniz zaman bunları değiştirebilirsiniz. Mesela SS pimi arduino uno için D10 pimidir. Bunun yanı sıra saat frekansı 4 e bölünmüştür. Siz daha yavaş bir frekans istediğinizde bunu setting kısmında belirtmeniz gerekir ya da   
  **SPI.setClockDivider(SPI\_CLOCK\_DIV2)** gibi bir kod ile değiştirmeniz gerekir. Bununla birlikte bit transferinin nereden başlayacağını da **SPI.setBitOrder(MSBFIRST)** ya da **LSBFIRST** belirtmeniz gerekir. Tabii ki mode kısmını da bu şekilde belirtmemiz gerek. **SPI.setDataMode(mode tablosuna göre yaz)** Genellikle alıcı cihazlar MODE0 ve MODE3 sinyal şeklini benimsiyormuş. En iyisi kullanacağınız alıcının DATASHEET dokümanına bakmanızdır.

**Mode resmini koy**

Mode konusunu anlat

Arduino ss tablosunu koy.

NOT: SS pini daima output olarak setup bölümünde tanımlanmalı. Tanımlanmadığı taktirde programdaki diğer kütüphaneler tarafından değiştirilebilir ve SPI haberleşmeniz çalışmaz olabilir.

Buraya kadar SPI ya giriş olarak bilgi vermeye çalıştım. Bundan sonra ise konumuz olan SPI serial memory entegrelerle nasıl bilgi alışverişi yapabiliriz onu göreceğiz. SPI kütüphanesine ait komutlar da çok fazla değildir. Bunu aşağıdaki örnek programlara bakarak siz de görebilirsiniz.   
  
Seri haberleşme kullanan bu tip hafıza entegreleri, SPI haberleşmesi kullandığından hızlı bir şekilde yazıp silme işlemlerinin yapılmasına olanak sağlıyor. Eskiden flash hafıza 100.000 kez silinip yazılırken teknolojinin gelişmesiyle 1.000.000 kez yazılıp silinmeye başladığı için de bu avantaj haline geldi.

Epromlar ile karşılaştırıldığında bazı dezavantajları var. Fakat yine de bazı SPI haberleşmeli entegreler bu dezavantajı ortadan kaldırmış görünüyor. Mesela 25LC512 (512Kbit) entegresi 1 kez yazdıktan sonra değişiklik yapmak istediğinizde, page erase denilen blok şeklinde silme işlemi yapmanızı istiyor. 25LC640A (64Kbit) entegresinde ise page erase yok. Aynı sayfaya tekrar tekrar yazabiliyoruz.   
EPROM da ise istediğimiz kadar silme işlemi yapmadan aynı hücreyi değiştirebiliyoruz. Genelde kullandığımız entegreler I2C haberleşmesi kullanıyorlar.

**I2C yazısının linkini koy**

I2C ile çalışan epromlar, Standard (100 kHz), Fast (400 kHz) and Fast−Plus (1 MHz) hızlarında haberleşiyorlar. Genelde 100khz ve 400khz kullanılıyor. SPI kullanan seri hafıza entegreleri ise 10Mhz hızına kadar haberleşebiliyor. Hatta bazıları 20Mhz e kadar haberleşebiliyormuş.

Başlıkta kütüphanesiz kullanım dedim. Fakat bu SPI kütüphanesini kullanmayacağım anlamına gelmez. Elbette SPI haberleşme komutlarını kullanmak için ona ihtiyaç var. (Ben SPI yı da kendim yazarım diyenler isterlerse onu da yazabilirler fakat vakit kaybı olur. )   
**Neden kütüphanesiz dedim.** Benzer entegreler genelde aynı komut yapısını kullansa da tek kütüphane hepsine yetmeyebilir. Bu kez kullandığınız entegreye ait kütüphane bulmanız gerekecektir.   
Örneklerde 25LC512 (512Kbit), AT25F512A(512Kbit),25LC640AT(64Kbit) entegreler kullanılmıştır.   
Programlar arasında örnek verilen 1 adet kütüphaneli program da var. (kendi programım değil örnek olsun diye ekledim)   
Bu 3 entegrenin yazma komutu okuma komutu aynı olsa da 1 tanesinde (25LC640AT) page erase komutu yok. Bir tanesinde tüm entegreyi sil komutu diğer entegreden farklı. Komut yazılışı aynı olsa bile kod farkı var. (2 adet 512Kbitlik entegrede). Yani tüm entegreler için benzer komutlar kullanılsa bile kodlar farklı olabiliyor.   
Ya da entegrenin kapasitesine göre kod dizilimin de adres bilgisi, 2 byte ya da 3 byte olabiliyor.

Bundan dolayı tüm entegrelere ait bir kütüphane yapsanız, bu kez de programlayacak hafıza bulamayabilirsiniz. Yazının ilerleyen bölümlerinde bunu daha iyi anlayacaksınız.

Çözüm nedir derseniz? Hangi entegreyi kullanıyorsanız o entegrenin kodlarını entegreye ait DATASHEET dosyasından elde etmektir. Datasheet te sadece kodlar yok. Her sayfa(page) kaç byte tan oluşuyor. Her sector(sektör) kaç byte tan oluşuyor. Kaç sektör var gibi bilgilerin yanı sıra maksimum kaç Mhz de iletişim kurmanız gerektiğini yazıyor. Mesela 25LC640AT 2Mhz iken 25LC512 10Mhz haberleşme imkanı veriyor. Bu bilgiler ışığında kolaylıkla haberleşebilirsiniz.

**Öncelikle SPI hazırlık kısmını görelim.**   
SPI için programın setup kısmında verilecek 2 komut var.

**SPI.begin ();** **SPI.beginTransaction(SPISettings(4000000, MSBFIRST, SPI\_MODE0));**

SPI begin satırı komut yazabilmeniz için kütüphaneyi aktif ediyor. 2. Satırda ise 4000000=4Mhz hızında haberleşmek istediğimizi belirtiyor.   
MSBFIRST dediğimiz kısım ise binary modunda kullanılan değerin en yüksek bitinden itibaren gönder demektir. Ayrıca Mode kısmını da ayarlamanız gerekiyor.  
Mesela 10001100 gibi bir sayımız var. Burada soldaki 1 biti en yüksek bittir. Yani MSB yi ifade eder. En sağdaki bit ise LSB yani en düşük biti ifade eder.   
Burada bit, byte kelimelerini bilmeyenler için açıklama yapmam gerektiğini düşünüyorum.   
Bit : dijital sistemlerde yapılan işlemlerde en küçük yapı taşı diyebiliriz. Ya 1 olur ya da 0 olur.   
Byte: 8 adet bitin bir araya gelmesiyle 1 byte oluşur. Buna örnek vermemiz gerekirse işlemciler 8 bit 32 bit 64 bit diye sınıflandırılır. Ayrıca yazdığımız ABC 123 gibi karakterler ise bilgisayarda ASCII kodları olarak bilinir. Bu ASCII kodları 8 bit ile ifade edilir. Yani her bir karakter 1 Byte lık yer kaplar.   
Hafıza entegreleri de her ne kadar 512Kbit olarak bit ile ifade edilse de yapıları 1 bytelık bilgilere göre çalışır. Yani kaç karakter alır bu entegre diyorsanız, 512K yı 8 bit sayısına bölmeniz gerekir.   
(Not: 512Kbit dediğimiz de 512000 bit demek değildir. Sayı sistemlerinde 1000 kat yerine 1024 kat büyür. Bundan dolayı 512\*1024=524288 bit yapar. )   
Bunu da 8 bit’e bölersek 512K=65536 byte (karakter) alır diyebiliriz.   
25LC512 gibi entegreler için söylersek datasheet de aksi belirtilmedikçe PAGE size(sayfa boyutu)=128byte, SECTOR size (Sektör boyutu)=16Kbyte dır.

SPI haberleşmede kullanılan diğer komutlar ise şu şekildedir.   
  
**digitalWrite(SS, LOW);** // CS Yani SS slave select pimi master tarafından LOW yapılarak haberleşme başlar.

**SPI.transfer (1);** // bilgi MOSI ucundan alıcının SI(slave input) ucuna MSB ilk gönderilecek şekilde Master cihazın sağladığı CLK palsi eşliğinde gönderilir.

**digitalWrite(SS, HIGH);** // daha sonrasında SS pimi High yapılarak işlem tamamlanır.  
   
Tabii ki burada gönderilecek kodlar alıcı cihazın istediği formatta gönderilecektir.   
Mesela Bu hafıza entegrelerinde Alıcı sistemin istediği komut şu şekildedir.  
**epromyazma sinyal ve okuma sinyal resimlerini koy**

Bununla birlikte yazmak için yazma korumasının da aşılması lazım.   
Bu tablo 25LC512 nin komut tablosudur. Buna göre yapmak istediğimiz işe göre kod yollarız.   
**Komut tablosunu koy**

Komut tablosuna bakarsanız write protect kısmının sadece donanımsal olarak yapılmadığını görürsünüz. Bu tabloda 6 kodu (Set the write enable latch (enable write operations)) Bir yazma işlemi yapılacağı zaman hafızadaki yazma kilidini kaldırır. Bazı komutlar bu olmadan çalışmaz. Yazma komutu ve page erase komutu gibi.   
Yukarıdaki sinyal resimlerine bakarsak 512Kbit entegrede 16 bitlik adresleme gerektiği görünüyor. Fakat denemelerimde ve kütüphane yazan kişilerin programlarında, komut ile adres arasında 1 adet 0 göndermek gerekecektir. Bu da 3 tane 8 bitlik adres gönderilmesi gerektiğini bize anlatıyor. Bu datasheet te nerede derseniz datasheet 16 bit yeter diyor fakat ben proteus üzerinde 16 bit adresle doğru kayıt yapamadım. Sonrasında kütüphaneli yapılan programında aynı şekilde araya 0 koyduğunu gördüm. Bu şunu gösteriyor. 512Kbit entegrede en fazla 65535=0xFFFF kayıt yapılabilmektedir. Bununla birlikte 1024Kbit entegrelerde ise daha 131072 byte= 0x20000 yani 3 byte adrese ihtiyaç var. Bundan dolayı 512K lık entegrede 3bytelık adrese sahip görünüyor. Datasheet 2 byte dese bile. Belki de sonradan uyumluluk olarak yapıldı. Bunun yanı sıra 64Kbit olan ise daha küçük olduğu için 2byte adres yeterli oluyor. **( Yani kullandığınız entegrenin boyutuna göre buradaki adres 2-3byte olabiliyor buna dikkat etmelisiniz. )**

Şimdi ise bir byte yazalım ve okuyalım.   
Yazma için komut satırları.

digitalWrite(SS, LOW); // Haberleşmeyi başlat  
SPI.transfer (6); // write enable latch (enable write operations) kod=6  
digitalWrite(SS, HIGH); // Haberleşmeyi kapat  
delay(10); // Süre hiç vermeseniz bile olur. İsteğe bağlı.   
  
digitalWrite(SS, LOW); // Haberleşmeyi başlat  
SPI.transfer (2); // write komutu 1-128byte yazabilirsiniz. 128byte=1 page  
SPI.transfer(addr >> 16); // 3.adres Byte adresin en büyük byte’ı MSB den dolayı  
SPI.transfer(addr >> 8); // 2. Adres byte  
SPI.transfer(addr); // 1. adres Byte   
SPI.transfer(yaz); // Yazılacak 1. değer  
 SPI.transfer(yaz+1); // yazılacak 2. Değer. Bu alt alta arttırılabilir.   
digitalWrite(SS, HIGH); // haberleşmeyi kapat.   
delay (100);  
Buraya kadar 2 bytelık bilgiyi yazdık. Addr değeri 0x000001 olduğunu düşünürsek yazılacak ilk değer hafıza entegresinin 1. Byte’ına 2. Değer 2. Byte’ına verilir. 128 byte içerisinde istediğiniz önceden yazılmamış bir yere yazdırabilirsiniz.

Okuma yapılacağı zaman yine benzer satırları göreceksiniz.   
digitalWrite(SS, LOW); // Haberleşmeyi başlat  
 SPI.transfer (3); // read byte komutu kod=3  
SPI.transfer(addr >> 16); // 3. Byte adresin en büyük byte’ı MSB den dolayı  
SPI.transfer(addr >> 8); // 2. byte   
SPI.transfer(addr); // 1. Byte   
uint8\_t result = SPI.transfer(0); // burada okumak istediğimiz bilgi için CLK palsini master devam //…………………………………………..ettirmek zorunda bundan dolayı bu 0 byte ını yollarız ve karşılığında //……………………………………………istediğimiz adresteki bilgiyi hafızadan okur ve master cihaza gönderir.   
uint8\_t result1 = SPI.transfer(0); // 2 bilgi yazdığımızdan 2. Bilgiyi de bu şekilde 0 göndererek cevabını //…………………………………………………….alırız.   
digitalWrite(SS, HIGH); // haberleşmeyi kapat  
Serial.print(" okunan deger : ");Serial.print(result,HEX);Serial.print(" , ");Serial.println(result1,HEX); // yukarıdaki satırda ise gelen değişkenleri ekrana yazdırırız.   
delay (10);

Yukarıdaki şekilde 2 byte yazdık 2 byte okuduk. Ekran görüntüsünde giden ve gelen bilgiler gösterilmektedir. ( Addr>>16 gibi satırlar görüyorsunuz bunlar verilen sayıyı binary olarak düşünüp sağ tarafa bit bit kaydığını düşünün. Bu işlemi yapar. Mesela 24 bit yani 3 byte sayınız var. En soldaki byte öğrenilmek istendiğinde 16 bit kaydırma yapılır ve en sağdaki byte olur bunu da bir değişkene aktarabiliriz ya da SPI transfer sadece 8 bit yolladığından en sağdaki byte’ı kullanacaktır. Bundan faydalanırız. En sağda kalan byte yollanacaktır. Burada addr>>16,addr>>8: 3. Ve 2. Byte ların en sağdaki byte’a kaydırılması kolayca transfer edilmesini sağlar. Son addr ise 3 byte lık veridir. Fakat SPI transfer sadece 8 bit yolladığından en sağdaki byte’ı kullanacaktır bundan dolayı 2. Ve 3. Byte lar işlem dışı kalacaktır. **Bu 2. Videoda görülmektedir.**   
**yazokuekran1.png koy**

SPI ile bir de Arduino.cc sitesinde örnek olarak bulunan dijital pot örneği programlar içerisinde bulunmaktadır.   
**Burada SPI haberleşme nasıl kullanılıyor.**

**Programın başında:**  
 const int slaveSelectPin = 10; //   
setup bölümünde  
pinMode (slaveSelectPin, OUTPUT); // set the slaveSelectPin as an output:  
 SPI.begin

**Loop bölümünde**  
digitalWrite(slaveSelectPin,LOW);  
SPI.transfer(address);  
SPI.transfer(value);  
digitalWrite(slaveSelectPin,HIGH);

Farkettiyseniz   
**SPI.beginTransaction(SPISettings(4000000, MSBFIRST, SPI\_MODE0));**

Gibi bir satır olmadığını görmüşsünüzdür. SPI haberleşme kütüphanesinde bazı başlangıç değerleri vardır. Bunlar ise burada kullanılmıştır. Bundan dolayı bu satırı kullanmamışlar. Fakat siz işi sağlama almak için datasheet dokümanına bakarak gerçek değerleri programınıza yerleştirmelisiniz.

**Öncelikle page boyutunda çalışmayı görelim.**   
Normalde adres 1 arttırılarak page boyutundan büyük bilgiler yazılabilir. Buraya kadar sorun yok. Fakat sorun 1 page boyutunda bilgiyi 1 komut satırıyla yazdırmaya kalktığınızda ortaya çıkıyor. Mesela 130 byte lık bir bilgiyi yazdırıyorsunuz. Bu fazlalık olan bilgi diğer page üzerine yazılacaktır derseniz, HATA yaparsınız. Maalesef bu bilgiler diğer PAGE bloğuna yazılmaz. Hatta PAGE başlangıç adresinden sonra bir adresten başladığınızda bilgiler yazılmaya başlanır ve PAGE sonuna gelindiğinde kalan veri varsa bunu PAGE başında yazmadığı yerlere yazar. Aşağıdaki örnek resimlerdeki gibi.   
Yazılan bilgi.128 Byte: “123456789098765432101234567890987654321  
01234567890987654321012345678909876543210123456789  
0987654321012345678909876543210Merhaba!"  
Bu bilgiler adres yanlış verildiğinde aşağıdaki resimlerdeki gibi görünür.   
**resim koy**

Yazılan bilgi 132 byte: "UUUU123456789098765432101234567890987654321  
0123456789098765432101234567890987654321  
01234567890987654321012345678909876543210Merhaba!"  
Bu şekilde bir komut satırında 132 karakter yazarsanız blok olarak yazdığınız için fazla olan karakterleri PAGE başlangıç yerine geri dönerek eski bilginin üzerine yazar. **Tabii ki bu durum yanlış durumdur.   
eprom data sayfa görüntülerini koy**

Bununla birlikte tek tek yazarak page sınırının dışına çıkabiliriz demiştik. Evet 1 komut satırında 128 byte yazmak yerine 128 kez yazmayı göze alarak 1 PAGE boyutundan büyük bilgiyi yazabiliriz.

**Resim koy**

**Not:** Bununla birlikte mesela 7. Byte tan sonra veriyi yazdınız elinizde 2 byte lık bilgi var bunları yazmak istediğinizde önceden yazılmamış 0. Ve 1. Adreslere yazabilirsiniz. PAGE içerisindeki diğer veriler zarar görmez.   
  
Sector ise (page) sayfalardan oluşur. Bu sayfalar blok olarak yazıldığı gibi tek tek de yazılabilir.  
Asıl problem şudur. Bir page içinde değişiklik yapılacak ise bu page içindeki veriler geçici olarak okunur ve okunan verilerde değişiklik yapılıp tekrar page erase yaptıktan sonra 128 byte olarak yerine konur.   
Şayet 1 hafıza entegresi almayı düşünürseniz datasheete bakın. En düşük page yapısı olmasına ve page erase olmasına bakın. Yoksa iş daha zor. Mesela AT25F512 de page erase yoktur. Page yapısı program diye bir komut ile yapılıyor. Fakat standarttan biraz farklı. Bu tip standart yapıdan farklı tipleri de seçmezseniz sizi zorlamayacaktır. Bu entegrede page erase yok, sector erase vardır. Bu da bir kerede 32K lık bilgiyi silmeniz demektir.   
Daha düşük page sayfası kullanan entegre sizin işinizi daha kolay görecektir. Bunun sebebi 128byte geri alıp değiştir ve tekrar yerine koy dediğinizde bu hem daha çabuk olur hem de hafızanızı zorlamayacaktır.   
**Fakat bu tip entegreyi kalıcı bir kod ile programlamak istiyorsanız sorun yok. Mesela bazı** programlarda fontlar arduino programına gömülür ya da resim bilgileri arduino içine gömülmeye çalışılır. Bu tip programların kalıcı ekran görüntülerinin imajını bu hafıza entegresine koyduğunuzda ekran değiştikçe, ekran görüntüsünü hafıza entegresinden çağırıp ekranı değiştirebilirsiniz. Bu tip hafıza entegrelerinin kullanımı, Mikrokontrol entegrenizin işini kolaylaştıracak ve daha fazla program hafızası bırakacaktır. SPI hızında okumak da size fazla zaman kaybettirmeyecektir. Mesela OLED displayi ele alalım. 128\*64 piksel olduğunu düşünelim ve 8 piksel bir byte olduğunu düşünürsek 4Mhz hızında SPI ile çalışırsak, ekranın tamamına ait bilginin okunması sadece 2,5ms sürmektedir. Bu da bizim fark edebileceğimiz bir hız değildir. Yani gayet hızlı okunuyor.   
Arduino ya da diğer mikrokontrolcüler de program ya da data hafızasından tasarruf etmiş olacaktır. Bununla birlikte birden fazla SPI cihazı da SS hariç aynı uçları kullanabileceğiniz için pim sorunu da olmayacaktır. Sadece her SPI cihaz için farklı bir SS pimi belirlemelisiniz.