BÖLÜM 12
SPI Arabirimi

Doç. Dr. Fatih Evran
Elektrik-Elektronik Mühendisliği
Düzce Üniversitesi
Email: fatihevran@duzce.edu.tr

SPI ve I2C Seri Arabirimleri

- SPI ve I2C PIC mikrodenetleyici üzerinde bulunan iki senkron seri arabirimidir.
- Hem SPI hem de I2C endüstride yaygın olarak kullanılır.
- En az üç bağlantı gerektirdiğinden SPI yapısı basittir.
  A. Çoğu veri aktarımı yarı çift yönlü (half-dublex) olmasına rağmen teknik olarak çift yönlüdür (full-duplex).
  B. PIC33'deki en yüksek çalışma frekansı 10 MHz'dir.
  C. Yüksek hızlı seri aktarıma sahiptir.
- I2C sadece iki bağlantı gerektirir.
  A. Veri aktarımı yarı çift yönlüdür.
  B. En yüksek çalışma frekansı 1 MHz'dir.
  C. 'Slave' cihazları seçmek için harici bir bağlantı kullanılmamaktadır.

Seri Çevresel Arabirimi (Serial Peripheral Interface – SPI)

Triple Axis Accelerometer - ADXL345

CAN – SPI adapter

Pressure Sensor - MS5803

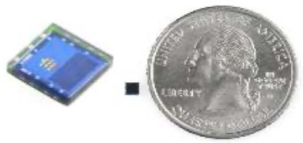
Barometric Pressure sensor

longwave infrared (LWIR) imager

Digital Temperature Sensor

DataFlash 16Mbit AT45DB161D

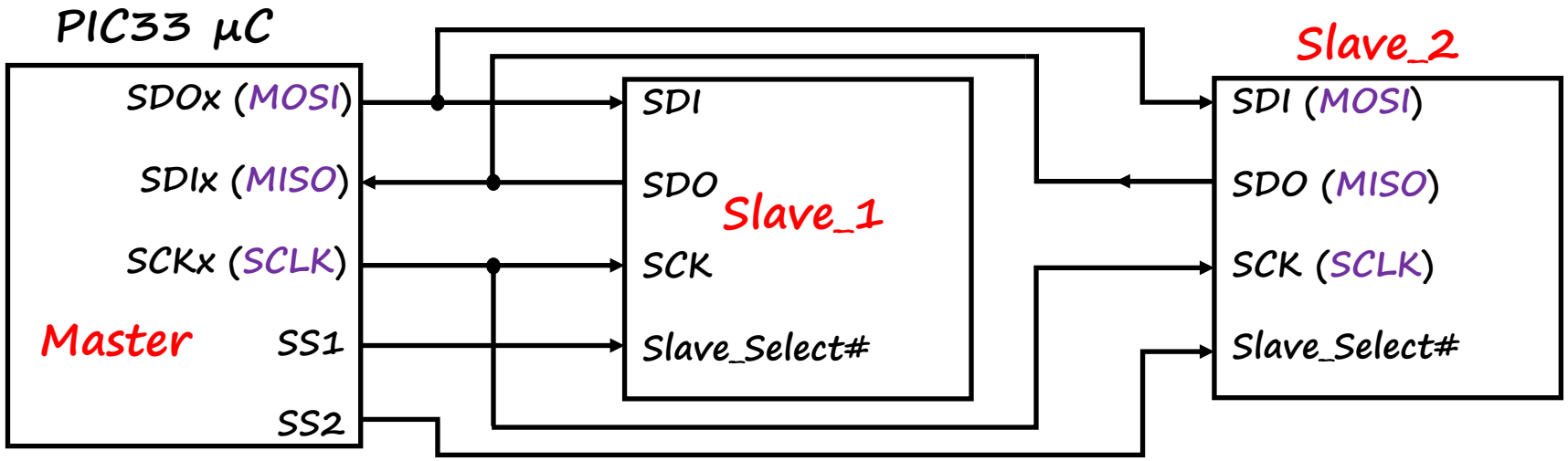
Tri-Axis Gyro Breakout - L3G4200D



Color Light Sensor

- Motorola tarafından geliştirilmiştir.
- Yazıcılar, kameralar, tarayıcılar gibi aygıtları bir masaüstü bilgisayara bağlamak için 90'lı yıllarda Seri Çevresel Arabirimi (SPI) kullanıldı, ancak günümüzde yerini USB arabirimine kaptırmıştır.
- Günümüzde ise SPI ekranlar, bellek kartları, sensörler gibi bazı uygulamalar için hala bir iletişim aracı olarak kullanılmaktadır.

Seri Çevresel Arabirimi (Serial Peripheral Interface-SPI)



SDO: Serial Data Out
SDI: Serial Data In
SCK: Shift Clock Input or Output

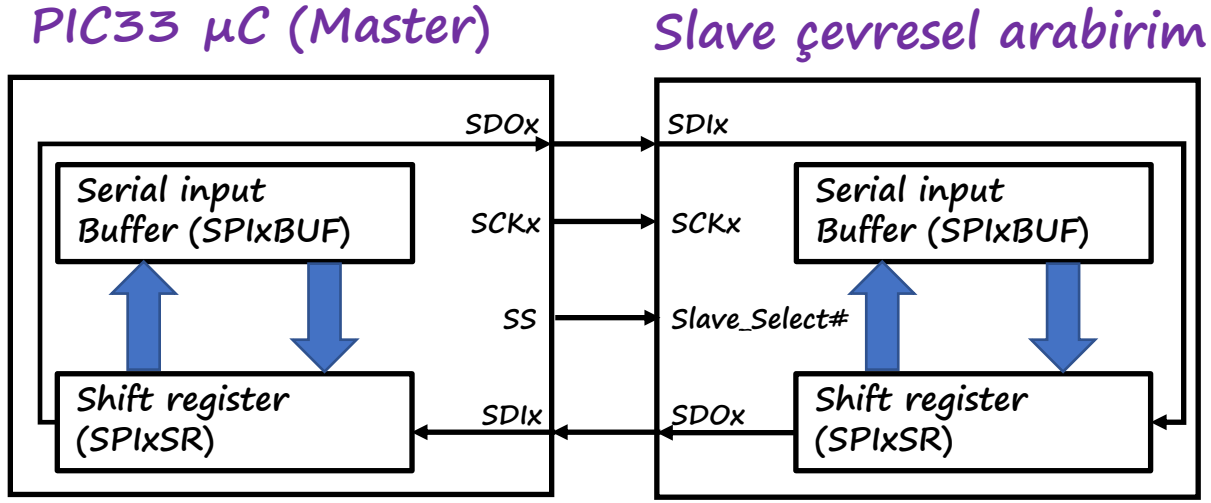
Farklı adlandırmalar da sıklıkla kullanılır:

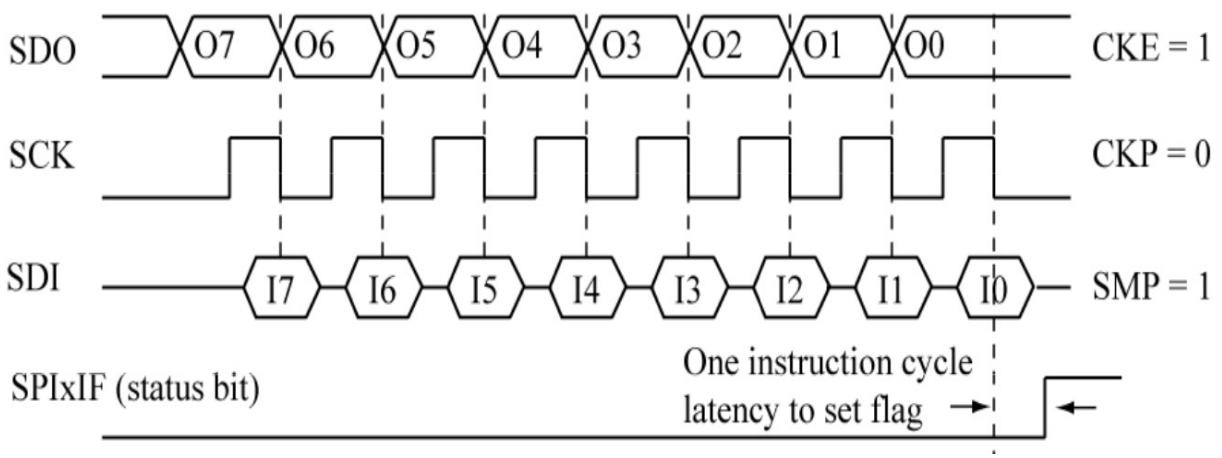
MOSI: Master Out Slave In-SDO
MISO: Master In Slave Out-SDI
SCLK: SPI Serial Clock-SCK

- SPI bir master/slave kurulumu kullanarak tam dupleks senkron modda çalışmaktadır (yani master ile slave arasında 8 bitlik veri eşzamanlı olarak iletilmektedir ve alınmaktadır).
- 'Master' aynı anda sadece bir 'slave' cihazı seçebilir. Birçok 'slave' cihaz üç durumlu çıkışlara sahiptir, bu nedenle seçilmedikleri zaman çıkış sinyalleri veri yolundan kesilmektedir.
- 'Master' veri gönderimini başlatmadan önce SSx (slave select-slave seçim) pinini düşük yani lojik '0' (ya da slave cihazın kullanım kılavuzuna bağlı olarak yüksek) durumunda tutarak haberleşmek istediği 'slave' cihazı seçmelidir ve SSx pini veri gönderimi-alımı süresi boyunca düşük (veya yüksek) kalmalıdır.
- 'Master' tarafından oluşturulan bir saat (SCK) yardımıyla, iki cihaz arasında ne zaman ve ne kadar hızlı veri alışverişi yapılacağı kontrol edilir. Veri gönderimi ise MSb bitinden LSb bitine doğru olmaktadır.

Master Veri gönderimi

- Master 'slave' cihazına veri göndermek için, 8 bitlik veriyi SPIxBUF yazmacına yazmaktadır. Bu veri ayrıca master'ın SPIxSR yazmacına otomatik olarak yazılmaktadır.
- SPIxBUF yazmacına veri yazılır yazılmaz, SCKx pininden sekiz saat darbesi gönderilir ve aynı zamanda veri bitleri SDOx üzerinden master SPIxSR'den slave SPIxSR'ye gönderilir, ve veri aktarımı sonunda master ve slave cihazlarının SPIxSR yazmaçlarının içeriği değiştirilmektedir.
- Bu veri iletiminin sonunda, SPIxIF bayrağı (SSP1IF (IFS0<10>), SSP2IF IFS2<1>) ve SPIRBF (SPIxSTAT<0>) biti veri aktarımın tamamlandığını göstermek için '1' olmaktadır.
- Mevcut veri tamamıyla iletilmeden önce SSPxBUF'a yeni bir verinin yazılmamasına dikkat edilmelidir, aksi takdirde bir 'yazma çarpışma- write collision, WCOL' hatası meydana gelecektir (SSPXCON1<7>).





PIC24 µC için SPIx Blok Şeması

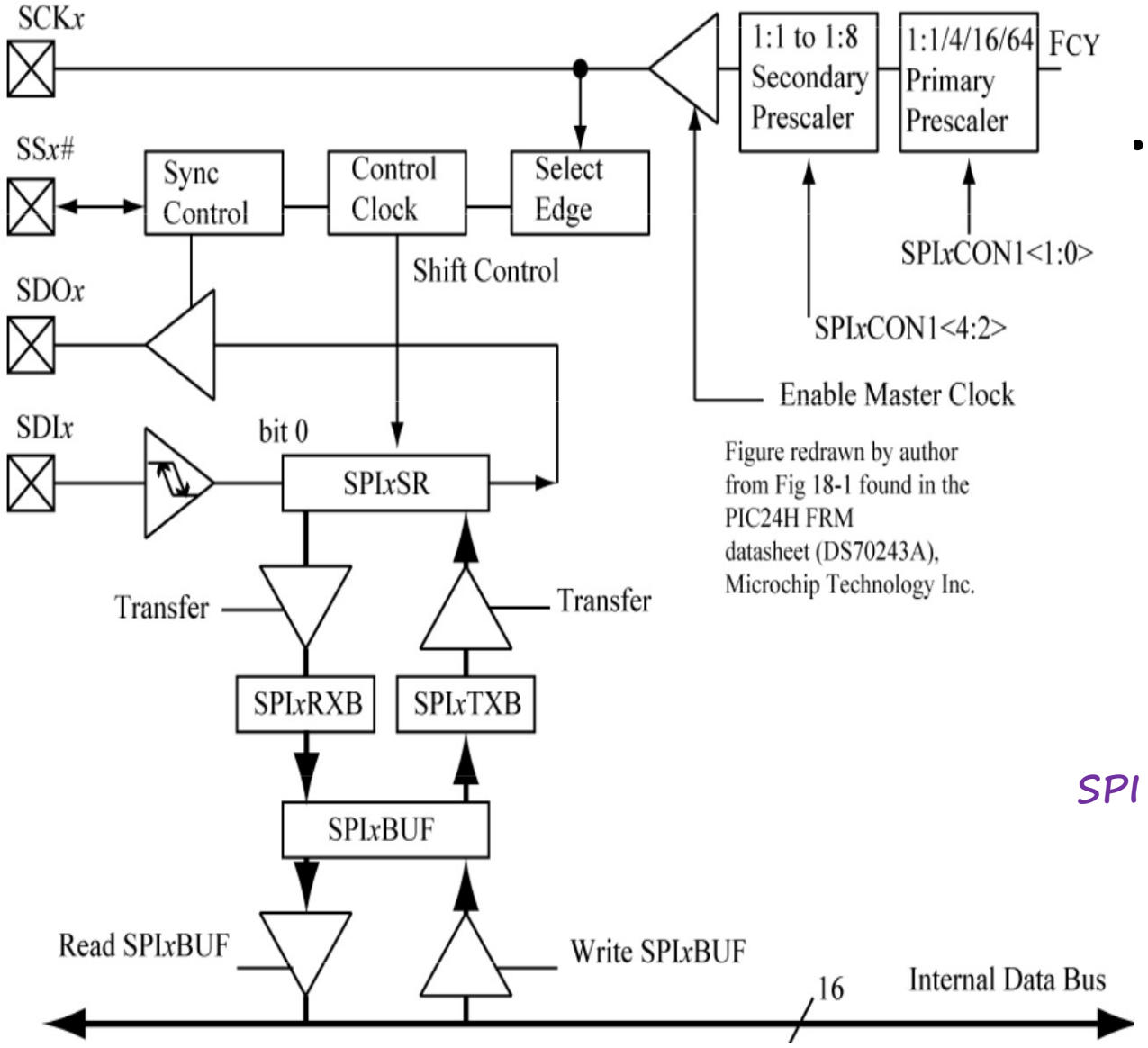


Figure redrawn by author from Fig 18-1 found in the PIC24H FRM datasheet (DS70243A), Microchip Technology Inc.

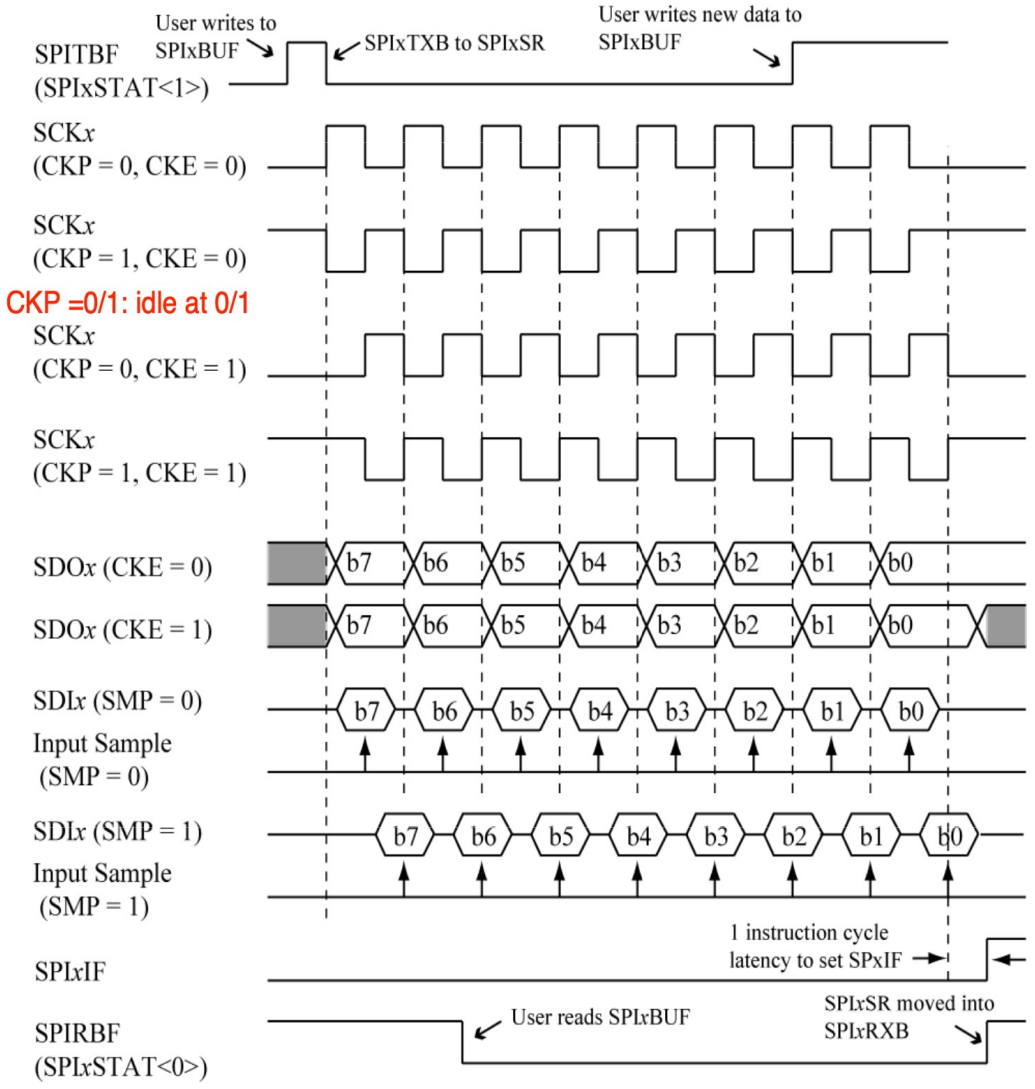
- PIC' de, SPI bağlantı noktasında saat frekansını ayarlamak için birincil bir ön bölücü (primary prescaler) ve ikinci bir ön bölücü (secondary prescaler) bulunmaktadır.
  - A. Birincil ön bölücü: SPIxCON1<1:0> bitleri ayarlanarak ön bölücünün değeri 1, 4, 16 veya 64 olmaktadır.
  - B. İkinci ön bölücü: SPIxCON1<4:2> bitleri ayarlanarak ön bölücünün değeri 1 ile 8 arasında olmaktadır.

SPI saat frekansı (FSCKx):

$$F_{SCKx} = \frac{F_{CY}}{SPIxCON1<1:0> \cdot SPIxCON1<4:2>}$$

SPI SAAT DALGA ŞEKİLLERİ

- CKE (SPIxCON1<8>, saat fazı ($\overline{CPHA}$)) yapılandırma biti, seri çıkış verisinin (SDO) saat kaynağının ilk ya da ikinci kenarında değişeceğini ayarlar (CKE=0 ise ilk kenarda veri değişir, CKE=1 ise ikinci kenarda veri değişir).
- CKP (SSPxCON1<6>) saat polaritesini (CPOL) seçer. CKP=1 ise veri transferi olmadığı durumda saat yolu '1' durumdadır, aksi takdirde saat yolu '0' durumundadır.
- SMP (SSPxSTAT<7>) biti, SDI seri veri girişinin saat periyodunun ortasında ya da sonunda örnekleneceğini belirler (SMP=0 ise örnekleme saat kaynağının ortasında, SMP=1 ise örnekleme saat kaynağının sonunda).



SPI Modları		
Standart SPI Mod	CKP (=CPOL)	CKE (=$\overline{CPHA}$)
00 (0)	0	1
01 (1)	0	0
10 (2)	1	1
11 (3)	1	0

- Çoğu aygıtta CKE=1 ve CKP=0 durumundadır.

SPI C Fonksiyonları

pic24_spi.c ve pic24_spi.h kütüphanelerinde bulunan iki temel fonksiyonu uygulamalarımızda kullanacağız

Overflow tespiti

```
void checkRxErrorSPI1() {
if (SPI1STATbits.SPIROV) {// If SPI buffer overflow flag is set
SPI1STATbits.SPIROV = 0; // clear the error
reportError("SPI1 Receive Overflow\n"); // Report the SPI overflow error
  }
}
```

Gönderme ve alma

```
uint16_t ioMasterSPI1(uint16_t u16_c) {
```

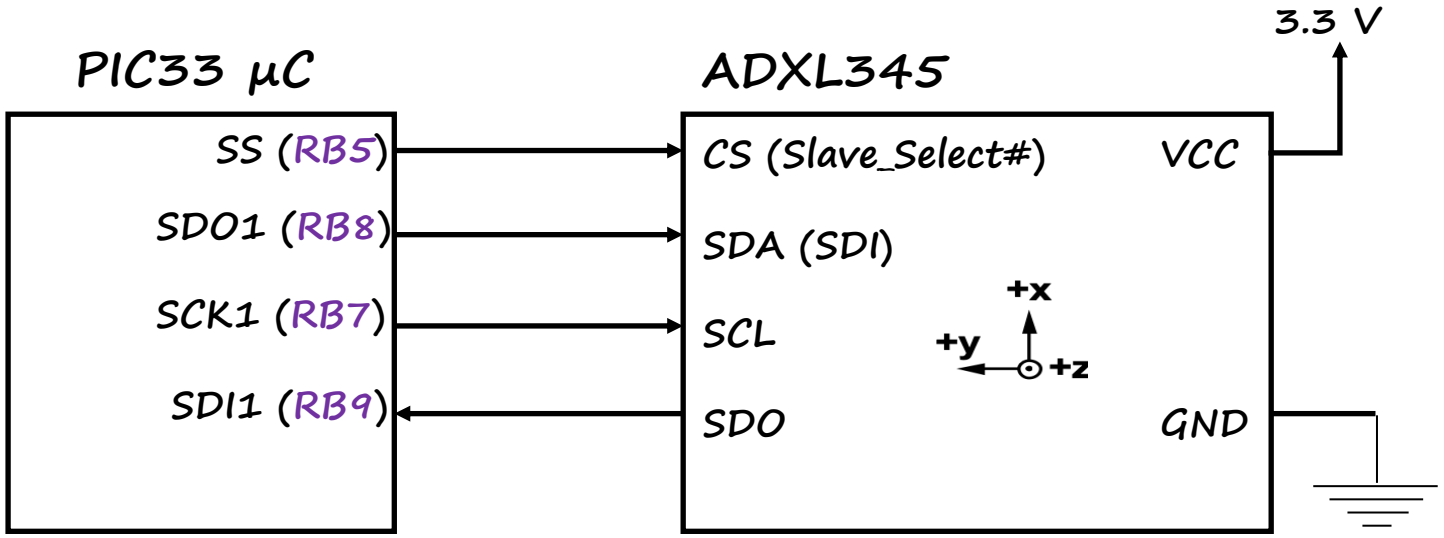
Veri gönderme

```
checkRxErrorSPI1();
_SPI1IF = 0; // clear interrupt flag since we are about to write new value
SPI1BUF = u16_c;
```

Veri alma

```
while (!_SPI1IF) { //wait for for SPI receiving interrupt
doHeartbeat();
};
return(SPI1BUF); // Return the received data from SPI buffer
}
```

ADXL345 – İvme Ölçer (Digital Accelerometer)



dsPIC33EP128GP502 işlemcisinde, SPI1 arayüz pinleri olan SCK1, SDO1 ve SDI1, sırasıyla RB7, RB8 ve RB9 pinleri ile aynı pin konumlarına sabitlenmiştir. SPI2 arayüz pinleri için ise yeniden eşlenebilir pinler kullanabilir

- ADXL345 3 eksen bir ivme ölçer sistemidir.
  - A. Farklı çalışma modlarına sahiptir
  - B. İvme ölçeri SPI portu üzerinden yapılandıracağız
  - C. CS pini düşük (lojik 0) olduğunda SPI iletişimi için ADXL345 seçilir
  - D. İvme ölçerin seçilebilir bir ölçüm aralığı ±2 g ila ±16 g' dır
  - E. Maksimum SPI saat kaynağı hızı 5 MHz olmaktadır.
- ADXL345 için çip seçimi olarak işlemcinin RB5 pinini kullancağız

ADXL345 Yapılandırma

Yükselen kenarda örnekleme (ikinci kenar) => SMP = 0 Cihaz etkinleştirme için düşük olmalıdır => CS = 0

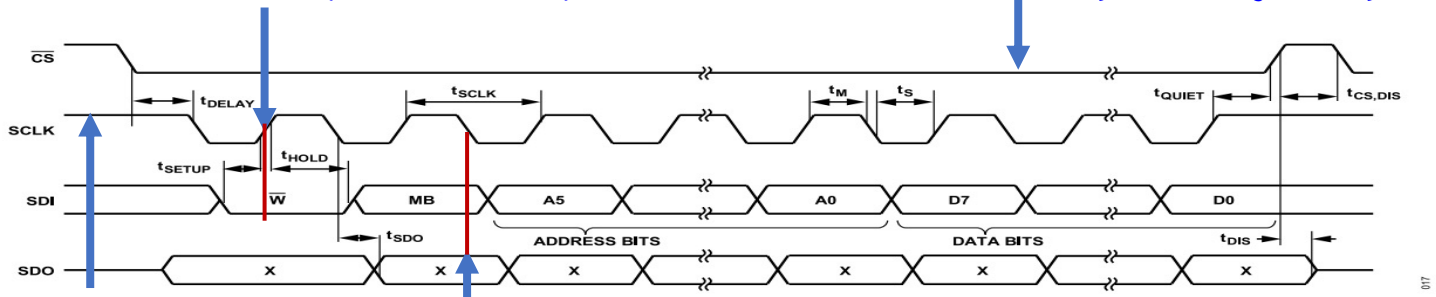


Figure 37. SPI 4-Wire Write

Boşta saat = Yüksek => CKP = 1

Düşen kenarda veri iletimi (ilk kenar) => CKE = 0

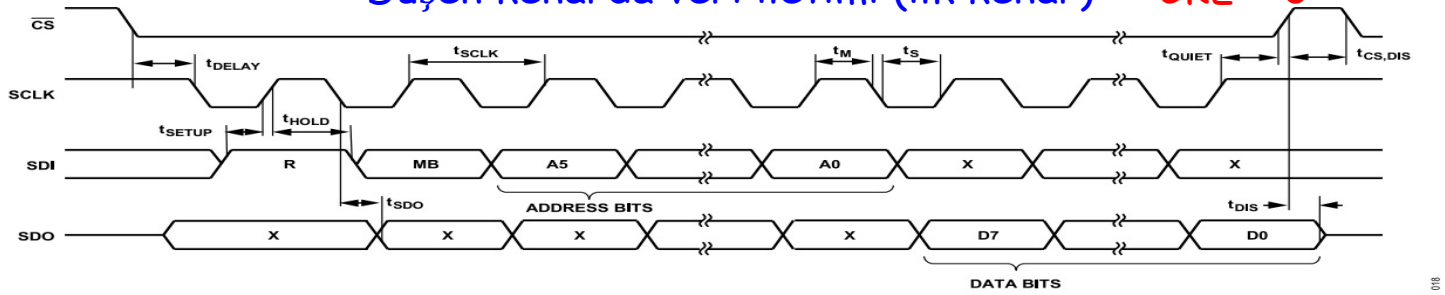


Figure 38. SPI 4-Wire Read

- Saat hattı boşta iken, saat hattın gerilimi '1' olduğu için CKP=1 olmaktadır.
- SDO üzerindeki veri değişimi, boş saat durumundan aktif duruma geçişte (düşen kenarda) gerçekleştiği için CKE=0 olmaktadır.
- SDI girişi saat periyodunun ortasında (yani yükselen kenarda) örneklendiği için SMP=0 olmaktadır.

ADXL345 Yapılandırma (devam)

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

DATA_FORMAT = 0x09 //Full_res (11 bits), ±4 g, 4-wire SPI mode, 3.90625 mg

Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

BW_RATE=0x0D //Output data rate=800

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

POWER_CTL = 0x08 //measurement mode

Address		Register		Reset value	Description
0x32	50	DATAX0	R	00000000	X-Axis Data 0.
0x33	51	DATAX1	R	00000000	X-Axis Data 1.
0x34	52	DATAY0	R	00000000	Y-Axis Data 0.
0x35	53	DATAY1	R	00000000	Y-Axis Data 1.
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0.
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1.

Veriler, DATAX, DATAY ve DATAZ saklayıcıları aracılığıyla okunur.

ADXL345 (1)

```
#define CONFIG_SLAVE_ENABLE() CONFIG_RB5_AS_DIG_OUTPUT()
#define SLAVE_ENABLE()        _LATB5 = 0  //high true assertion
#define SLAVE_DISABLE()       _LATB5 = 1 //use RB5 to select ADXL345
void configSPI1(void) {
  //spi clock = 70MHz/(5*4) = 70MHz/20 = 3.5MHz
  SPI1CON1 = SEC_PRESCAL_5_1 |      //1:1 secondary prescale
             PRI_PRESCAL_4_1 |      //4:1 primary prescale
             CLK_POL_ACTIVE_LOW | //idle state for clock is a high level,
                                   //active is low level  (CKP = 1)
             SPI_CKE_OFF  | //out changes inactive to active (CKE=0)
             SPI_MODE8_ON | //8-bit mode
             MASTER_ENABLE_ON;      //master mode
  CONFIG_SLAVE_ENABLE();        //chip select for ADXL345
  SLAVE_DISABLE();              //disable the chip select
  SPI1STATbits.SPIEN = 1;   //enable SPI mode and configure SDO1, SDI1 and SCK1
}
```

ADXL345 (2)

ADXL345' i yapılandırın: yapılandırma adresini yazın, ardından yapılandırma verisini yazın

```
void writeConfigADXL345(uint8_t address,uint8_t value) {
  SLAVE_ENABLE();           //assert chipselect
  ioMasterSPI1(address);    //config address
  ioMasterSPI1(value);      //config value
  SLAVE_DISABLE();
}
```

ADXL345' in yapılandırma saklayıcısına yazar

ADXL345' ten veri oku: adresi yaz, ardından ADXL345' ten veri oku

```
int16_t readADXL345(uint8_t address) {
  uint8_t temp;
  address = 0x80 | address;// reading mode
  SLAVE_ENABLE();         //assert chipselect
  ioMasterSPI1(address);    //address
  temp = ioMasterSPI1(0x00); //read Byte
  SLAVE_DISABLE();
  return(temp);
}
```

ADXL345' ten 8 bitlik değeri okur

Verileri geri almak için sahte veriler gönderin

Tek bir 8 bitlik değer olarak döndürülen eksen verileri

ADXL345 (3)

```
int main (void) {
  int16_t i16_temp;
  int xhi, xlo, yhi, ylo, zhi, zlo;
  long Xaccumulate, Yaccumulate, Zaccumulate;
  uint8_t i;
  configBasic(HELLO_MSG);
  configSPI1();
  //DATA_FORMAT = 0x09
  //config address,config value,Full_res (11 bits),-+4g, 3.90625 mg
  writeConfigADXL345(DATA_FORMAT,0x09);
  //POWER_CTL = 0x08
  writeConfigADXL345(POWER_CTL,0x08); //config address, config value, measurement mode
   //BW_RATE = 0x0d
  writeConfigADXL345(BW_RATE,0x0d);   //config address, config value, measurement mode
  device_id=readADXL345(DEVID); //device_id=229
  printf("DEVID = %d\r\n",device_id);
  while (1)
  {
   …
  }
}
```

ADXL345 (4)

```
while (1)
  {
    Xaccumulate = Yaccumulate = Zaccumulate = 0;
    for (i=0; i<16; i++)
        { // Read sequentially 16 times then get an average
            xlo=readADXL345 (DATAX0);     // DATAX0 is the first of 6 bytes
            xhi=readADXL345 (DATAX1);     // read character
            ylo=readADXL345 (DATAY0);     // read character
            yhi=readADXL345 (DATAY1);     // read character
            zlo=readADXL345 (DATAZ0);     // read character
            zhi=readADXL345 (DATAZ1);     // read character
            Xaccumulate += ((xhi<<8) | xlo);
            Yaccumulate += ((yhi<<8) | ylo);
            Zaccumulate += ((zhi<<8) | zlo);
        }
    Xaccumulate=Xaccumulate>>4; //Xaccumulate = Xaccumulate + (xhi*256 + xlo)
    Yaccumulate=Yaccumulate>>4;
    Zaccumulate=Zaccumulate>>4;
    printf("X = %ld (%.2f g), Y=%ld (%.2f g), Z=%ld (%.2f g)
       \r\n",Xaccumulate,Xaccumulate * 0.00390625,Yaccumulate,Yaccumulate *
       0.00390625,Zaccumulate,Zaccumulate * 0.00390625);
      DELAY_MS(1000);
  }
```

Program Çıktısı