Serial Peripheral Interface

Der vorliegende Artikel ist ein Leitfaden zum einbinden von Sensoren, die eine SPI-Schnittstelle bieten. Das Serial Peripheral Interface (SPI) ist ein serielles Bus-System mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können. Die Datenkommunikation ist digital und erfolgt nach einem standardisierten Modell. Weiter Beispiele für serielle Bussysteme sind I^2C, CAN-Bus oder USB. Die Verkabelung von SPI Schaltungen braucht drei bis vier Verbindungen (CS, SPC, SDI, SDO) und weitere Schaltungen können nicht angehängt werden, sondern müssen unabhängig verkabelt werden. Im Gegensatz dazu biete I^2C eine Schnittstelle auf Basis von zwei Kabeln und erlaubt die Adressierung von 112 Knoten pro Bus. Das SPI-Interface erlaubt deutlich höhere Datenraten.

Funktionsweise

SPI verwendet zur Kommunikation zwei Steuer- und ein bis zwei Datenleitungen. Die Kommunikation erfolgt digital in Datenpaketen. Unten stehende Grafik illustriert die Funktionen der einzelnen Leitungen. SDI (Serial Data Input) und SDO (Serial Data Output) sind die beiden Datenleitungen, CS (Clock) und SPC (Serial Port Clock) die Steuerleitungen.

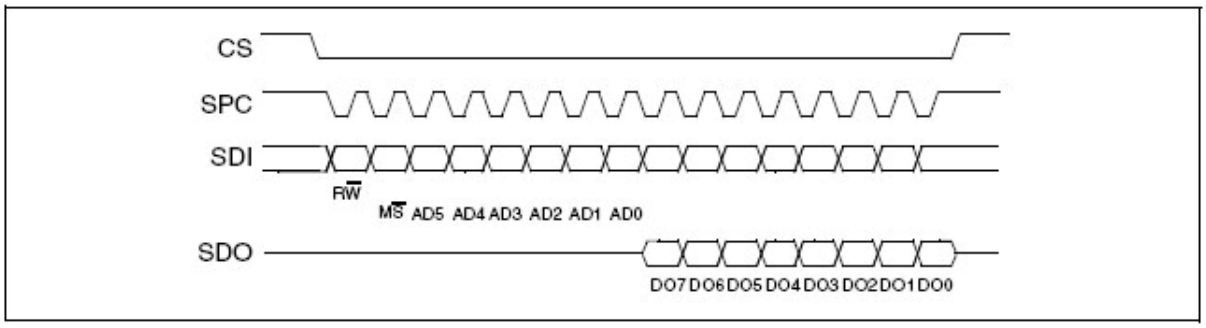

Abb1. Read & write protocol

CS und *SPC:* Die Steuerleitungen dienen der Synchronisation der Datenpakete. CS dient der Synchronisation eines Datenpaketes und ist vor und nach dem Senden eines Paketes HIGH und während der Datenübertragung LOW. SPC dient der Synchronisation der einzelnen Bits und geht von HIGH zu LOW und wieder zurück zu HIGH.

SDI und *SDO*: Die Datenleitungen dienen dem Lesen oder Schreiben von Daten. Oft ist es möglich, eine Leitung für das Lesen und Schreiben zu verwenden und damit die Anzahl der benötigten Verbindungen auf drei zu reduzieren.

Protokoll

Das erste Byte dient der Adressierung, Byte 2 und alle weiteren Bytes enthalten die Daten.

Byte 1:
Bit 0: (RW) Signalisiert ob geschrieben (LOW) oder gelesen (HIGH) werden soll.
Bit 1: (MS)
Bit 2-7: (AD5 – AD0) Registeradresse
Byte 2:
Bit 8-15: (DO7 – DO0) Daten

Byte 3 bis …:
Bit 16-…: Daten im multi byte reading

SPI Read

Untenstehende Grafiken verdeutlichen den Lesevorgang beim SPI-Protokoll. Folgende Schritte sind nötig um aus einem bestimmten Register zu lesen:

CS auf LOW um den Lesevorgang zu starten.
Byte 1 mit der Adresse des Registers auf SDI senden. Bit 0 ist HIGH da gelesen werden soll, Bit 1 LOW und Bit 2 bis 7 enthalten die Adresse. Für jedes Bit wird die Datenleitung entweder auf LOW oder HIGH gesetzt und mit SPC HIGH, dann SPC LOW gesendet.
Byte 2 enthält den Wert des Registers. Jedes Bit muss einzeln gelesen werden. Lesen ob SDO HIGH oder LOW, mit SPC HIGH, dann SPC LOW nächstes Bit lesen.
SPC und CS auf HIGH um den Lesevorgang zu beenden.

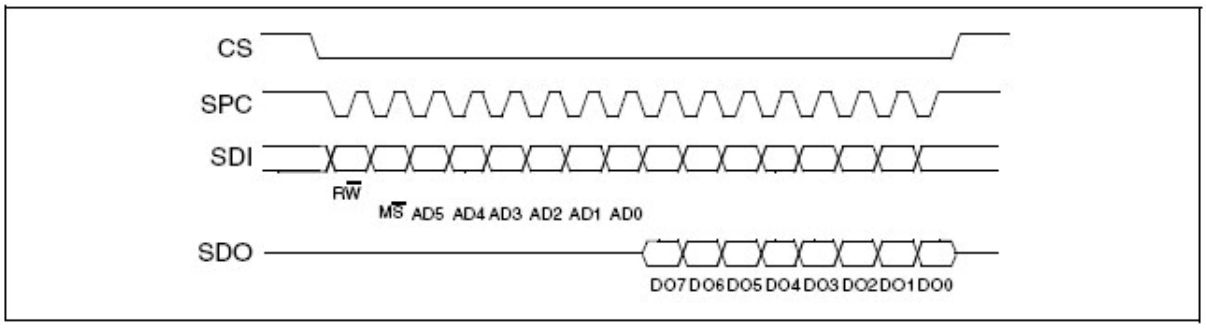

Abb2. SPI Read protocol

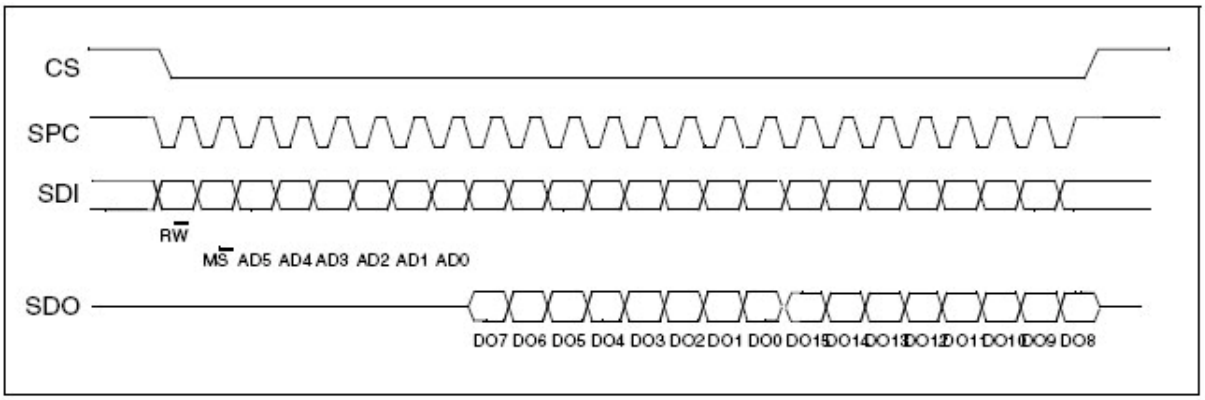

Abb3. Multiple bytes SPI Read Protocol (2 bytes example)

SPI Write

Untenstehende Grafiken verdeutlichen den Schreibvorgang beim SPI-Protokoll. Folgende Schritte sind nötig um in ein Register zu schreiben:

CS auf LOW um den Schreibvorgang zu starten.
Byte 1 mit der Adresse des Registers auf SDI senden. Bit 0 ist LOW da geschrieben werden soll, Bit 1 LOW und Bit 2 bis 7 enthalten die Adresse. Für jedes Bit wird die Datenleitung entweder auf LOW oder HIGH gesetzt und mit SPC HIGH, dann SPC LOW gesendet.

Byte 2 enthält den Wert, der in das Register geschrieben werden soll. Für jedes Bit wird die Datenleitung entweder auf LOW oder HIGH gesetzt und mit SPC HIGH, dann SPC LOW gesendet. SPC und CS auf HIGH um den Schreibvorgang zu beenden.

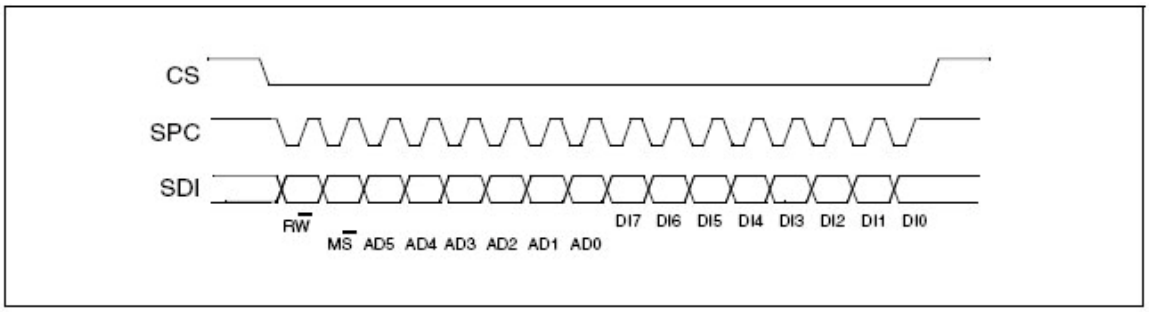

Abb1. SPI Write protocol

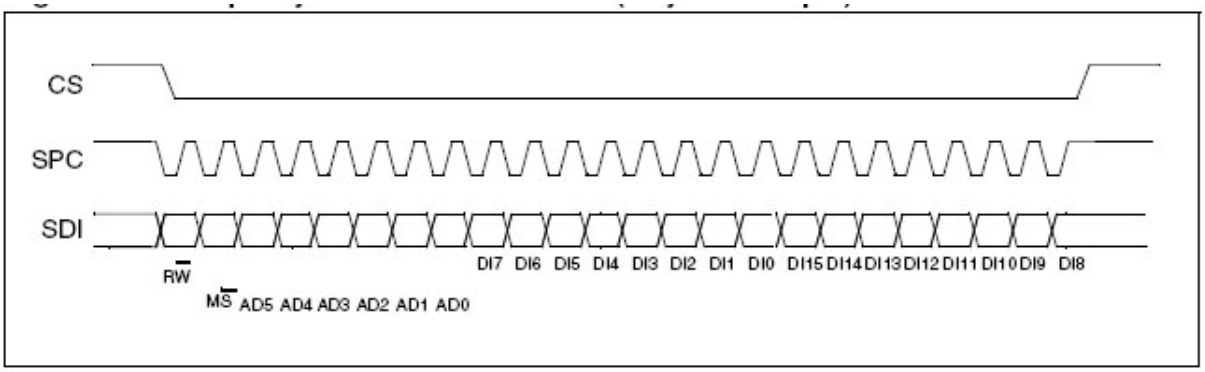

Abb1. SPI Read in 3-wires mode

Wiring/Arduino Example

```
//
// SPI_write

void SPI_write (byte registerByte, byte commandByte) {

 digitalWrite(CSB, LOW); //turn on device

 // 6 bits register address
 for (int i=5; i>=0; i--){

  // write bit
  if (int(1<<i & registerByte) == 0) digitalWrite(DATAOUT, LOW); else digitalWrite(DATAOUT, HIGH);
  // cycle clock
  digitalWrite(SCKCLOCK, HIGH);
  digitalWrite(SCKCLOCK, LOW);

 }
```

```
 digitalWrite(DATAOUT, HIGH); // Indicate that we are Writing
 digitalWrite(SCKCLOCK, HIGH);
 digitalWrite(SCKCLOCK, LOW);
 digitalWrite(DATAOUT, LOW); // LOW
 digitalWrite(SCKCLOCK, HIGH);
 digitalWrite(SCKCLOCK, LOW);


 // write 8 bits register content
 for (int i=7; i>=0; i--){

   // write bit
   if (int(1<<i & commandByte) == 0) digitalWrite(DATAOUT, LOW); else digitalWrite(DATAOUT,
HIGH);
   // cycle clock
   digitalWrite(SCKCLOCK, HIGH);
   digitalWrite(SCKCLOCK, LOW);

 }


 digitalWrite(CSB, HIGH); //turn off device

}



//
// SPI_read8

int SPI_read8 (byte registerByte) {

 digitalWrite(CSB, LOW); //turn on device

 // 6 bits register address
 for (int i=5; i>=0; i--){

   // write bit
   if (int(1<<i & registerByte) == 0) digitalWrite(DATAOUT, LOW); else digitalWrite(DATAOUT, HIGH);
   // cycle clock
   digitalWrite(SCKCLOCK, HIGH);
   digitalWrite(SCKCLOCK, LOW);

 }

 digitalWrite(DATAOUT, LOW); // Indicate that we are Reading
```

```
 digitalWrite(SCKCLOCK, HIGH);
 digitalWrite(SCKCLOCK, LOW);
 digitalWrite(SCKCLOCK, HIGH);
 digitalWrite(SCKCLOCK, LOW);

 // read 8 bits register content
 int returnValue = 0;
 for (int i=7; i>=0; i--){

  // read bit
  returnValue += digitalRead (DATAIN) << i;
  // cycle clock
  digitalWrite(SCKCLOCK, HIGH);
  digitalWrite(SCKCLOCK, LOW);

 }


 digitalWrite(CSB, HIGH); //turn off device

 return returnValue;

}



//
// SPI_read16

int SPI_read16 (byte registerByte) {

 digitalWrite(CSB, LOW); //turn on device

 // 6 bits register address
 for (int i=5; i>=0; i--){

  // write bit
  if (int(1<<i & registerByte) == 0) digitalWrite(DATAOUT, LOW); else digitalWrite(DATAOUT, HIGH);
  // cycle clock
  digitalWrite(SCKCLOCK, HIGH);
  digitalWrite(SCKCLOCK, LOW);

 }

 digitalWrite(DATAOUT, LOW); // Indicate that we are Reading
 digitalWrite(SCKCLOCK, HIGH);
```

```
  digitalWrite(SCKCLOCK, LOW);
  digitalWrite(SCKCLOCK, HIGH);
  digitalWrite(SCKCLOCK, LOW);


  // read 8 bits register content
  int returnValue = 0;
  for (int i=15; i>=0; i--){

    // read bit
    returnValue += digitalRead (DATAIN) << i;
    // cycle clock
    digitalWrite(SCKCLOCK, HIGH);
    digitalWrite(SCKCLOCK, LOW);

  }

  digitalWrite(CSB, HIGH); //turn off device

  return returnValue;

}
```