Arduino verstehen und anwenden

Teil 24: SPI – Der serielle periphere Interface-Bus

Nachdem in den letzten beiden Beiträgen der I²C-Bus vorgestellt wurde, sollen in diesem Artikel die Einsatzmöglichkeiten der seriellen Kommunikation mittels SPI (Serial Peripheral Interface) genauer erläutert werden. Da die Leistungsfähigkeit der seriellen Busse immer weiter verbessert wird, tritt der Geschwindigkeitsvorteil der parallelen Datenübertragung zunehmend in den Hintergrund. Die Taktfrequenzen von SPI-Bausteinen können bis zu einigen Megahertz und mehr betragen. Im Umfeld der Arduino-Anwendungen gibt es daher viele Einsatzmöglichkeiten, bei denen eine serielle Übertragung vollkommen ausreicht.

Der SPI-Bus ist nicht auf die klassischen Anwendungen in der Messtechnik oder auf die Ansteuerung von Peripheriebausteinen beschränkt. Auch im Audiobereich oder in der Unterhaltungselektronik findet er in zunehmendem Maße Anwendung.

Neben der Bezeichnung SPI, die von Motorola eingeführt wurde, findet sich häufig auch der von National Semiconductor geprägte Begriff „Microwire". Hinter beiden Bezeichnungen verbirgt sich aber dasselbe Funktionsprinzip.

Dass der Einsatz von seriellen Übertragungen auf dem Vormarsch ist, zeigt auch die Popularität anderer serieller Bussysteme wie I²C, das in den letzten Artikeln vorgestellt wurde, oder das inzwischen zum Universalstandard avancierte USB-System.

Aktuell sind Hunderte von SPI-Bausteinen verfügbar. Ihre Anwendungen reichen vom einfachen EEPROM über DACs und ADCs bis hin zu LED-Controllern, Temperatursensoren oder sogar Audio-Mischern. In diesem Artikel sollen aber die Bausteine im Vordergrund stehen, die besonders häufig im Arduino-Umfeld zu finden sind.

Der SPI-Bus

Eine klassische serielle Schnittstelle mit Tx- und Rx-Leitungen wird als „asynchron" (im Gegensatz zu „synchron") bezeichnet. Es existiert keine Steuerleitung, die kontrolliert, wann Daten gesendet werden. Deshalb kommt es zu Problemen, wenn beide Seiten nicht mit exakt der gleichen Taktfrequenz arbeiten.

Wenn über die Schnittstelle verbundene Systeme mit etwas unterschiedlichen Takten laufen, sind Übertragungsfehler vorprogrammiert.

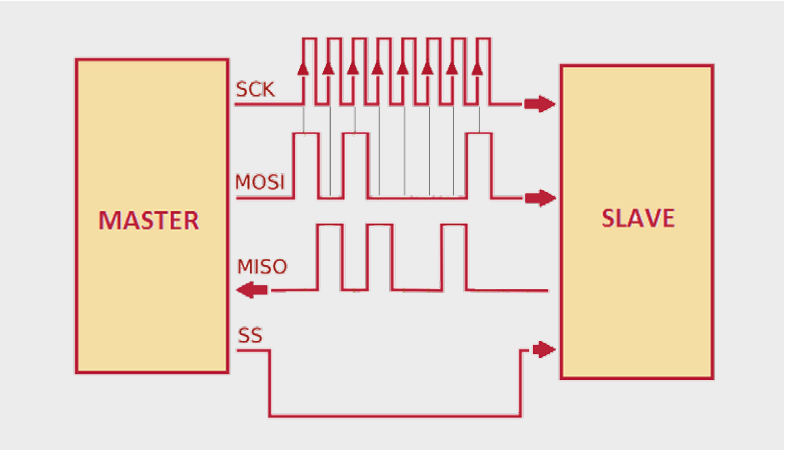

Bild 1: Datenkommunikation via SPI

Um dieses Problem zu entschärfen, fügen asynchrone serielle Verbindungen zusätzliche Start- und Stopp-Bits zu jedem Byte hinzu. Diese erlauben dem Empfänger eine Synchronisierung auf den Datenstrom. Beide Seiten müssen sich im Voraus auf die Übertragungsgeschwindigkeit (z. B. 9600 Bits pro Sekunde) einigen. Geringfügige Unterschiede in der Übertragungsrate sind dann unproblematisch, da der Empfänger bei jedem Byte neu synchronisiert.

Die asynchrone serielle Datenübertragung funktioniert prinzipiell recht problemlos und hat weite Verbreitung gefunden. Durch die zusätzlichen Start- und Stopp-Bits muss jedoch ein vergleichsweise großer Overhead in Kauf genommen werden. Zudem ist eine vergleichsweise komplexe Hardware erforderlich, um Daten senden und empfangen zu können.

Wie den meisten Anwendern der seriellen Schnittstelle bekannt ist, werden nur unverständliche Daten empfangen („*++'''##aaccc$$$..."), wenn beide Seiten nicht auf die gleiche Geschwindigkeit eingestellt sind. Das liegt daran, dass der Empfänger die Bits zu falschen Zeiten abtastet und deshalb auch die falschen Pegel detektiert.

Ein synchroner Datenbus dagegen verwendet separate Leitungen für Daten und Übertragungstakt. Die beiden Seiten müssen also nicht mehr über einen eigenen perfekt synchronen Takt verfügen. Der übertragene Takt teilt dem Empfänger mit hoher Präzision mit, wann er die Bits auf der Datenleitung abtasten soll. Dies kann die ansteigende oder abfallende Flanke des Taktsignals sein. Wenn der Empfänger die Flanke erkennt, tastet er unverzüglich die Datenleitung ab, um das nächste Bit zu lesen (siehe Bild 1). Weil der Takt zusammen mit den Daten gesendet wird, ist die Angabe einer Taktfrequenz (der sogenannten „Baudrate") hier nicht erforderlich.

Ein Grund für die große Beliebtheit des SPI-Busses ist, dass die empfangende Hardware lediglich aus einem einfachen Schieberegister

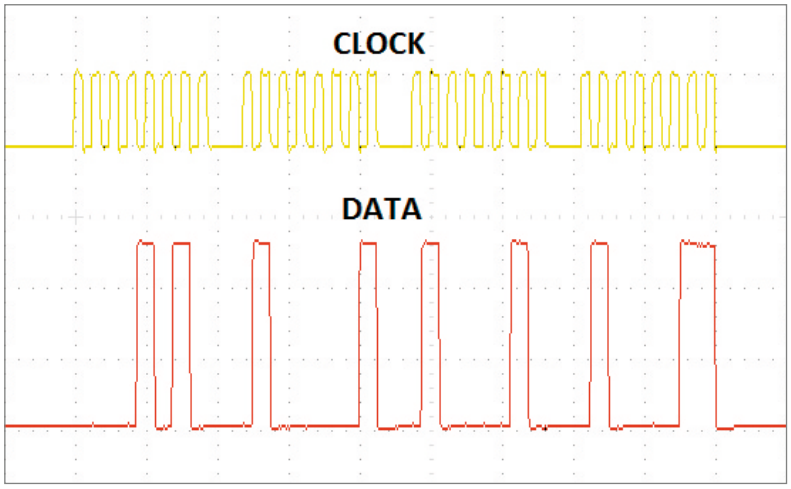

Bild 2: Ein am SPI-Bus gemessenes Oszillogramm zeigt CLOCK und DATA.

bestehen kann. Dieses stellt eine viel einfachere und billigere Lösung dar als die hochintegrierten UART-Bausteine (Universal Asynchronous Receiver/Transmitter) für asynchrone serielle Schnittstellen. Entsprechende Anwendungen mit dem bekannten 74HC595-Schieberegister finden sich weiter unten.

Der SPI-Bus benötigt zwei Steuerleitungen:
- Chip Select (CS) und
- SCLK (Serial Clock)

sowie zwei Datenleitungen:
- Serial Data In (SDI) und
- Serial Data Out (SDO)

Die beiden Datenleitungen werden häufig auch als
- MOSI (Master Out Slave In) und
- MISO (Master In Slave Out)

bezeichnet. Die Chip-Select-Leitung trägt oft auch den Namen SS für Slave-Select.

Mit der Chip-Select-Leitung kann jeweils der gewünschte Peripheriebaustein im Bussystem ausgewählt werden. Dieser Pin ist low-aktiv, d. h., im Ruhezustand sind die SDO-Leitungen hochohmig und damit vom Datenbus abgekoppelt. Der Master kann so bestimmen, mit welchem Peripheriegerät er kommuniziert. Die Clock-Leitung SCK dient als Taktsignal und wird unabhängig vom Selektionszustand zugeführt. Sie dient der Synchronisierung der Datenkommunikation. Bild 2 zeigt, wie ein SPI-Telegramm und das zugehörige Clock-Signal in der Praxis aussehen.

Bei den meisten SPI-Bausteinen sind alle vier Leitungen vorhanden. Es kommt allerdings auch vor, dass eine Leitung entfallen kann. Ein Peripherieelement, das nicht konfiguriert werden muss oder kann, benötigt beispielsweise keine Eingangsleitung, sondern nur einen Datenausgang. Sobald es angewählt wird, beginnt es mit dem Versenden von Daten. Bei einigen ADCs wurde daher auf die überflüssige SDI-Leitung verzichtet. Daneben gibt es auch Bausteine, die keine Datenausgangsleitung benötigen. LCD-Controller beispielsweise müssen lediglich Daten empfangen aber nichts zurückmelden. In diesem Fall kann also die SDO-Leitung entfallen.

Protokoll und Adressierung

Da es für SPI keinen offiziellen Standard gibt, ist es häufig erforderlich, die Datenblätter der verwendeten Bausteine zu Rate zu ziehen. Wichtig sind vor allem die erlaubten Taktfrequenzen und die Art der gültigen Flanken.

In der Praxis haben sich mehrere Betriebsarten durchgesetzt. Diese müssen jeweils korrekt vom Controller umgesetzt werden.

Hier kommt wieder der besondere Vorteil des Arduino-Umfelds zum Tragen. Für sehr viele Bausteine wurden passende Bibliotheken entwickelt und kostenlos zum Download im Internet zur Verfügung gestellt. Der Anwender muss sich also nicht mehr mit den einzelnen Betriebsarten auseinandersetzen. Es muss lediglich die erforderliche Bibliothek installiert werden und der entsprechende Baustein ist problemlos ansprechbar.

Die verschiedenen Peripherietypen

Nachfolgend sollen nun einige Peripheriebausteine vorgestellt werden. Die diversen Komponenten lassen sich in folgende Hauptkategorien unterteilen:
- Signalwandler (ADC und DAC)
- Speicher (EEPROM und FLASH)
- Real Time Clocks (RTC)
- Sensoren (Temperatur, Druck etc.)
- Sonstige (digitale Potentiometer, LCD-Controller, UART, CAN-Controller, USB-Controller, Signalmischer, digital gesteuerte Verstärker)

In den ersten drei Kategorien Wandler, Speicher und RTCs sind die meisten Bausteine zu finden. Komponenten aus den letzten beiden Gruppen werden dagegen eher selten eingesetzt.

Verschiedene Hersteller bieten Bausteine mit unterschiedlichsten Auflösungen, Taktfrequenzen und Kanälen an. Die Bitauflösungen reichen von 8, 10, 12 bis hin zu 24 Bit. Die Taktfrequenzen liegen bei 30 Kilosamples pro Sekunde bis hin zu 600 Kilosamples pro Sekunde. Bekanntermaßen besitzt der Arduino zwar sechs interne ADCs, allerdings weisen diese lediglich eine Auflösung von 10 Bit auf. Möchte man genauer messen, so ist ein SPI-Wandler häufig das Mittel der Wahl.

Anders sieht es bei den DACs aus. Die kleineren Arduinos (UNO oder MICRO) verfügen nicht über die Möglichkeit, konstante Analogsignale zu erzeugen. Hier muss man im Bedarfsfall auf die Pulsweitenmodulation ausweichen. Benötigt man jedoch ein echtes Analogsignal, muss man auf einen externen Baustein zurückgreifen und diesen z. B. via SPI ansteuern.

Die Speicherbausteine sind überwiegend EEPROM-Varianten. Es sind zwar auch einige SPI-Flash-Speicher verfügbar, aber nur vergleichsweise wenige. Der Spezialfall der SD- und microSD-Karten wird in einem eigenen Abschnitt behandelt.

Die Speicherkapazitäten der EEPROMs reichen von einigen Bit bis hin zu mehreren Kilo-Bit. Bei den Speichergeschwindigkeiten sind Taktfrequenzen von bis zu 3 MHz verfügbar.

Real Time Clocks (RTCs) bieten sich für eine serielle Kommunikation an, da hier keine großen Datenmengen übertragen werden. Auch bei den RTCs existieren verschiedene Varianten. Viele RTCs decken einen weiten Spannungsversorgungsbereich ab. Einige arbeiten schon mit einer Betriebsspannung ab 2.0 Volt. Sie können daher leicht mit einer Akku-Pufferung versehen werden. Weitere Details dazu finden sich im letzten Artikel zu dieser Serie.

Neben den Standardfunktionen einer 24-h-Uhr bieten einige Vertreter zusätzlich Funktionen wie Kalender oder Alarmeinstellungen.

Die Gruppe der SPI-Sensoren ist bislang noch recht überschaubar. Hier sind nur einige Temperatur- und Drucksensoren verfügbar. In diesem Bereich wird aufgrund der meist niedrigen Datenübertragungsrate häufig der I²C-Bus eingesetzt. Ob die Anzahl der SPI-Sensoren in Zukunft zunimmt, muss sich noch zeigen. Sind Messungen mit extrem hohen Datenraten nicht erforderlich, können viele Sensortypen, wie zum Beispiel für Windgeschwindigkeit, Luftfeuchtigkeit, Temperatur, Gaskonzentration, Lichtstärke usw., auch mit langsamen Schnittstellen erfasst werden.

Die SPI-Schnittstelle auf dem Arduino

Auf dem Arduino ist die SPI-Schnittstelle zweimal vorhanden. Einmal ist sie an der Standard-Pinleiste links oben verfügbar. Hier belegt sie die Digitalpins D10 bis D13. Standardmäßig liegt SS auf Pin D10. Aber natürlich können für den Slave-Select-Pin auch andere I/O-Ports verwendet werden. In den unten stehenden Beispielen kommt so etwa D08 als SS-Pin zum Einsatz. Zusätzlich sind die Leitungen aber auch am sogenannten ICSP-Stecker (In Circuit Serial Programming) aufgelegt (siehe Bild 3). Damit ist der Arduino bestens für den Anschluss von Bausteinen aus der SPI-Familie gerüstet. Der Hintergrund dieser doppelten Verfügbarkeit ist, dass der Controller des Arduino auch über die SPI-Schnittstelle programmiert werden kann. Weitere Details dazu finden sich im nachfolgenden Text.

Die SPI-Familie

In der folgenden Tabelle sind einige der wichtigsten SPI-Bausteine zusammengefasst:

Baustein	Funktion
74HC595	Schieberegister
MAX7219	LED-Display-Treiber
AD8403	Digitalpotentiometer
RC522	RFID-Transponder
nRF24L01	Radio Module
MCP4912	10 Bit DAC

Natürlich ist diese Tabelle bei Weitem nicht vollständig. Es wurden vielmehr die am häufigsten im Arduino-Umfeld eingesetzten Bausteine ausgewählt.

Neben den in der Bauteilliste aufgeführten SPI-Elementen gibt es eine Reihe weiterer Komponenten, die das SPI-Interface lediglich für das Einstellen der Konfigurationsregister benutzen. Die eigentliche Datenübertragung erfolgt aber über ein anderes Protokoll. So ermöglichen CAN- oder USB-Controller mit SPI-Schnittstelle dem Arduino die Kommunikation mit diesen Bausteinen.

Auch die serielle Anbindung von LC-Displays stellt eine interessante Alternative dar, da man hier sowohl wertvolle IO-Pins als auch viele Leitungen einsparen kann. Das untenstehende Praxisbeispiel verdeutlicht diese Variante.

Schieberegister als SPI-Emulator

Wie bereits erwähnt, kann er SPI-Bus bereits mit einem einfachen Schieberegister emuliert werden.

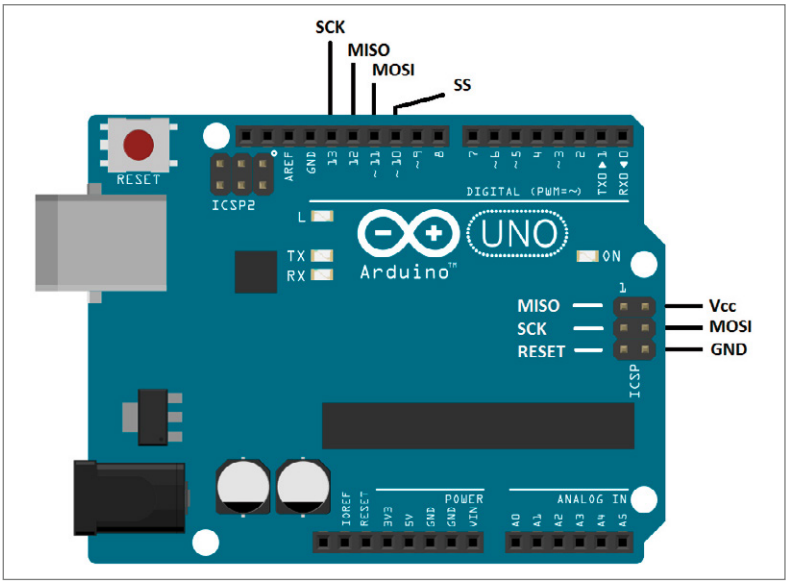

Bild 3: Die SPI-Schnittstelle auf dem Arduino

Hierbei handelt es sich formal gesehen zwar nicht um eine echte SPI-Bus-Übertragung, dennoch kann die Methode sehr nutzbringend eingesetzt werden.

Zum Einsatz kommt der bekannte Schieberegisterbaustein 74HC595. Dieser ist im Handel für wenige Cents erhältlich.

Der folgende Sketch zeigt, wie der Baustein über den SPI-Bus angesteuert wird. Zunächst muss natürlich mit

```
#include <SPI.h>
```

die SPI-Library eingebunden werden.

Dann erfolgt die Festlegung der benötigten Pins über entsprechende „define"-Anweisungen. Im Setup wird lediglich der Select-Pin als Ausgang definiert und der SPI-Bus gestartet.

In der Hauptschleife des Programms werden dann via

```
SPI.transfer(1<<j);
```

die einzelnen Bits in das Schieberegister übertragen. Der Bitshift-Operator „<<" sorgt dabei dafür, dass die übertragene Eins immer jeweils um eine Stelle verschoben wird. Dies hat zur Folge, dass am Ausgang des Schieberegisters ein Lauflicht-Effekt entsteht. Dieser kann über eine LED-Reihe sichtbar gemacht werden.

```
//SPI_74HC595.ino

#include <SPI.h>

#define SS 8              // 74HC595 ST_CP -> D08
#define CLOCK 13          // 74HC595 SH_CP -> D13
#define MOSI 11           // 74HC595 DS -> D11
#define MISO 12           // unused

void setup()
{ pinMode(SS, OUTPUT); //  // SS as output
  SPI.begin();
}

void loop()
{ for (int j = 0; j <= 7; j++)
  { digitalWrite(SS, LOW);
    SPI.transfer(1<<j);       // send Byte to 74HC595
    digitalWrite(SS, HIGH);
    delay(100);
  }
}
```

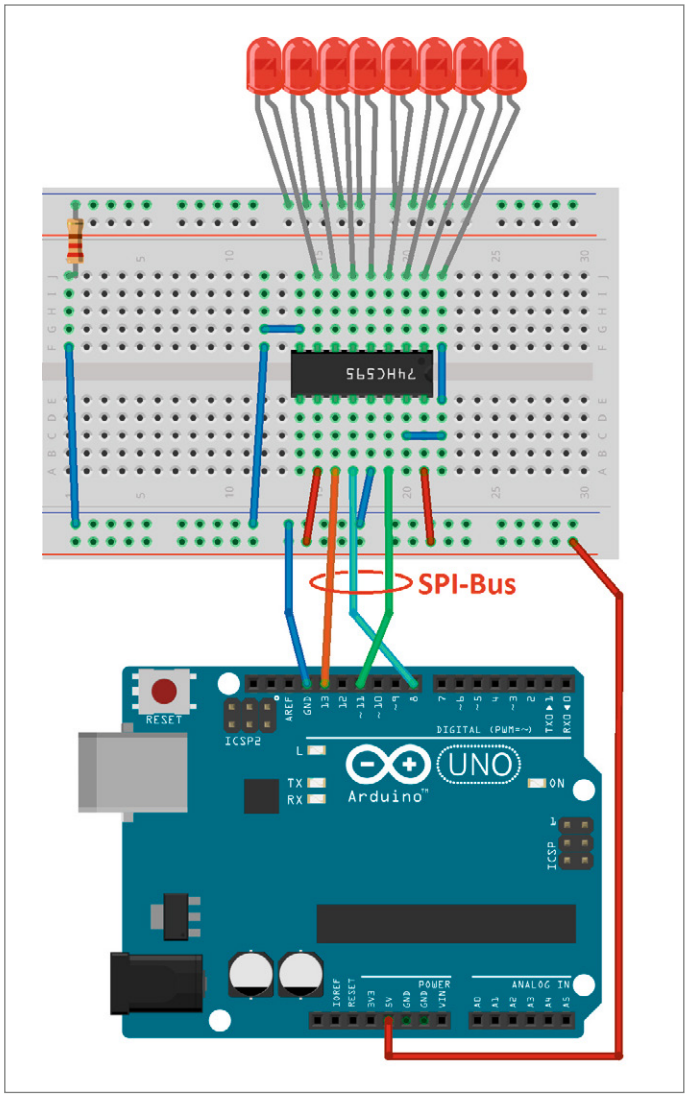

Bild 4: Der 74HC595 als SPI-Porterweiterung

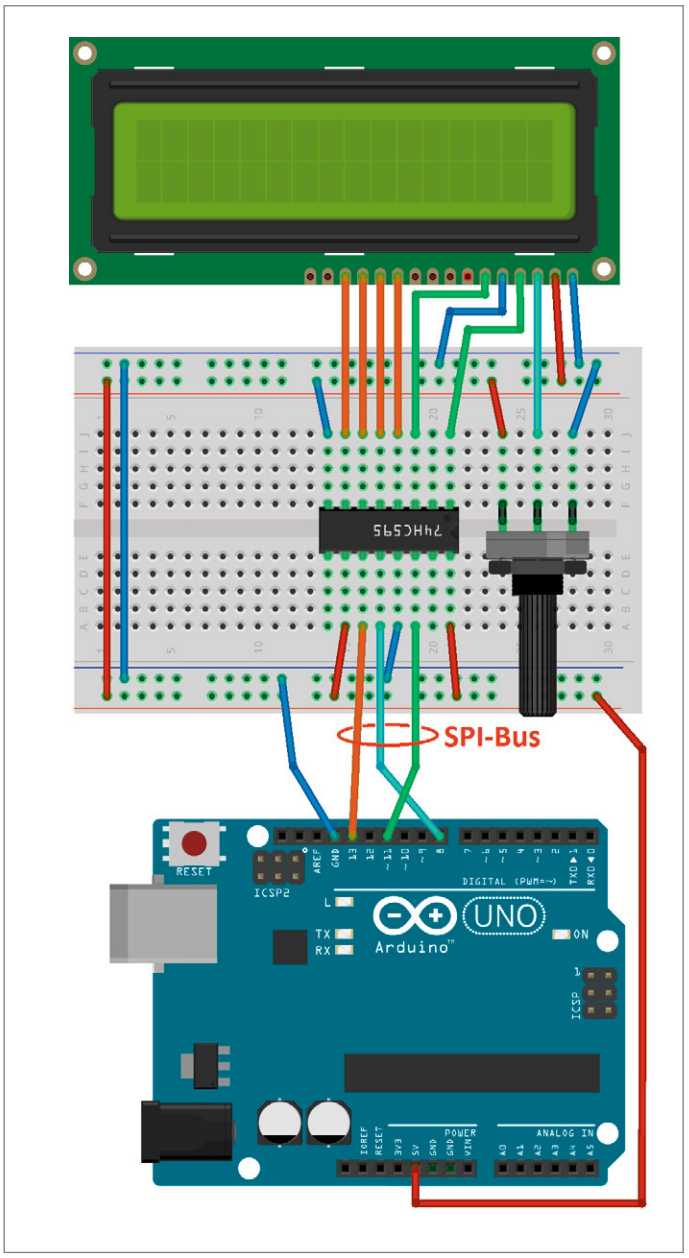

Bild 5: LCD-Display mit SPI-Interface: Der 74HCT595 macht's möglich!

Der dazu passende Aufbau ist in Bild 4 dargestellt.

Praxisanwendung: LCD-Display am SPI-Bus

Eine weitere sehr praktische Anwendung des Schieberegisters 74HC595 ist die Ansteuerung von LCD-Einheiten. Diese benötigen bei direktem Anschluss an den Arduino bis zu sechs Leitungen. Bei Verwendung der SPI-Übertragung kommt man mit lediglich drei Pins aus. Zudem können am Bus dann auch noch weitere Elemente angeschlossen werden, ohne dass zusätzlich Pins beansprucht werden. Es ist lediglich eine weitere Select-Leitung pro Busteilnehmer erforderlich.

Die zu dieser Anwendung passende Bibliothek kann unter

https://github.com/omersiar/ShiftedLCD

kostenlos aus dem Internet geladen werden.

Nach der Installation der Library zaubert der folgende Sketch die Nachricht „Hello ELV“ auf das Display.

```
//SPI_LCD.ino

#include <ShiftedLCD.h>
#include <SPI.h>

// initialize the library with the number of the sspin
// i. e. latch pin of the 74HC595
LiquidCrystal lcd(8);

void setup()
{ // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
}

void loop()
{ lcd.print("Hello ELV!");
  while(1);
}
```

Bild 5 zeigt wieder den dazugehörigen Aufbau. Das Potentiometer dient wie üblich zur Einstellung des Display-Kontrastes.

LED-Treiber MAX7219 für 7-Segment-Anzeigen und DOT-Matrizen

Der MAX7219-LED-Treiber spart sowohl Prozessor-Pins als auch Rechenzeit. Mit ihm können mit nur vier Arduino-Pins bis zu 64 LEDs angesteuert werden. Darüber hinaus können mehrere 7219-Chips für noch größere Displays miteinander verkettet werden.

Der MAX7219 verfügt über ein 4-Pin-SPI-Interface mit Clock, MOSI, Chip Select und Ground. Der Baustein gehört damit zur Kategorie der reinen Slave-Devices. Eine MISO-Leitung ist also nicht vorhanden, da der Baustein nur Daten empfängt und keine Rückmeldungen sendet. Folgende Tabelle fasst die wichtigsten Daten des Bausteins zusammen:

Versorgungsspannung	4,0–5,5 V
Stromaufnahme	330 mA
Maximaler Segmentstrom	40 mA
Scan-Rate	500–1300 Hz (800 Hz typ.)
Takt	Max. 10 MHz

Eine geeignete Library zur Ansteuerung der Maxim-ICs findet sich unter:

https://code.google.com/archive/p/arduino-maxmatrix-library/downloads

Passende Beispiele etwa für die Darstellung von Laufschriften oder Tickern sind ebenfalls auf dieser Web-Page unter Wikis vorhanden.

Der Anschluss des Treiberbausteins ist ebenfalls sehr einfach. Lediglich drei Pins sind mit dem Arduino zu verbinden:

MAX7219_data → Arduino D12
MAX7219_load → Arduino D10
MAX7219_clock → Arduino D11

Die Zeilen- und Spaltenpins der Matrix werden entsprechend ihrer Polarität an die Segment- bzw. Digit-Anschlüsse geschaltet. Schließlich ist am Pin I_set noch ein einzelner Widerstand von ca. 10 kΩ erforderlich, über den die LED-Ströme eingestellt werden. Damit können nahezu beliebig große Displays aufgebaut werden. Der Arduino wird kaum mehr belastet, da nur noch einzelne Kommandos über den SPI-Bus gesendet werden müssen. Zudem bleiben noch ausreichend Pins frei, um externe Sensoren oder andere Peripherie zu betreiben. So können auch großflächige Anzeigen, etwa für Werbezwecke oder Sportveranstaltungen, leicht realisiert werden.

Weitere Details zu Ansteuerung von Matrizen finden sich im Artikel „Einsatz und Anwendung von LED-Matrizen" im ELV Journal 3/2016 (Juni/Juli 2016).

Funkstrecken mit dem nRF24L01

Möchte man Daten über größere Entfernungen übertragen, hat aber keine geeigneten Kabel oder Kabelkanäle zu Verfügung, dann kann eine Funkbrücke Abhilfe schaffen. Auch bei der Heimautomatisierung können moderne Funkmodule sehr nützlich sein. Häufig will man hier keine zusätzlichen Kabel verlegen, insbesondere wenn keine Leerrohre mehr verfügbar sind.

Ein sehr beliebtes Modul, das für diese Zwecke häufig zum Einsatz kommt, ist das nRF24L01-Funkmodul. Diese kleinen Platinen bieten umfangreiche integrierte Funktionen. Adressierung, Sende- und Empfangscodierung etc. sind bereits im Chip integriert, deshalb sind zur Ansteuerung nur noch wenige Befehle erforderlich. Durch die Verwendung des SPI-Busses stellt auch die Hardwareseite keine große Herausforderung dar.

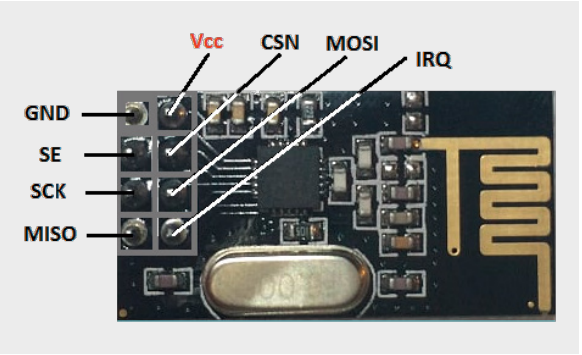

Bild 6: Das Funkmodul nRF24L01 weist ebenfalls ein SPI-Interface auf.

Bild 6 zeigt ein solches Modul. An den Bezeichnungen MISO, MOSI und SCK ist das SPI-Interface leicht zu erkennen.

Die Module senden auf dem 2,4-GHz-Band. Man kann bis zu 128 Kanäle wählen, um Funk-Kollisionen mit anderen Netzen zu vermeiden. Jedem Chip kann eine 40-Bit-Adresse zugewiesen werden. Beim Absetzen eines Funkpaketes sorgen die Chips selbst dafür, dass die Übertragung erfolgreich ist, d. h., sie senden ein Funkpaket so oft neu, bis die Gegenstelle den erfolgreichen Empfang mit einem Acknowlegde bestätigt. Die Ansteuerung und der Einsatz dieser nützlichen Helfer wird in einem späteren Artikel genauer betrachtet.

Auch SD-Karten werden über SPI angesprochen

SD-Karten kann man ebenfalls über ein entsprechendes Modul mit Kartenslot mit dem Arduino verbinden. Hierfür gib es spezielle Shields, aber auch z. B. das Ethernet-Shield bietet bereits einen Steckplatz für eine SD-Karte. Dadurch kann man sehr leicht und kostengünstig den Speicher des Arduino mittels SD-Karten auf eine Größe von mehreren Gigabyte erweitern.

Alle SD-Card-Reader für den Arduino kommunizieren via SPI-Protokoll mit dem Arduino. Daher kann man alle Module oder Shields mit der Standard-SD-Library des Arduino ansprechen.

Die Reader und der Arduino werden über die bereits bekannten Pins verbunden. Das bedeutet natürlich wieder, dass auch bei Verwendung des SPI-Headers statt der digitalen I/O-Pins diese I/Os trotzdem nicht anderweitig verwendet werden können. Das Signal SS beziehungsweise CS ist am Arduino UNO immer digital Pin D10. Das Pin-out des SPI-Headers des UNO sieht also so aus:

- 1: MISO, oder digital 12
- 2: Ground
- 3: SCK, oder digital 13
- 4: MOSI, oder digital 11
- 5: Reset
- 6: + 5 V

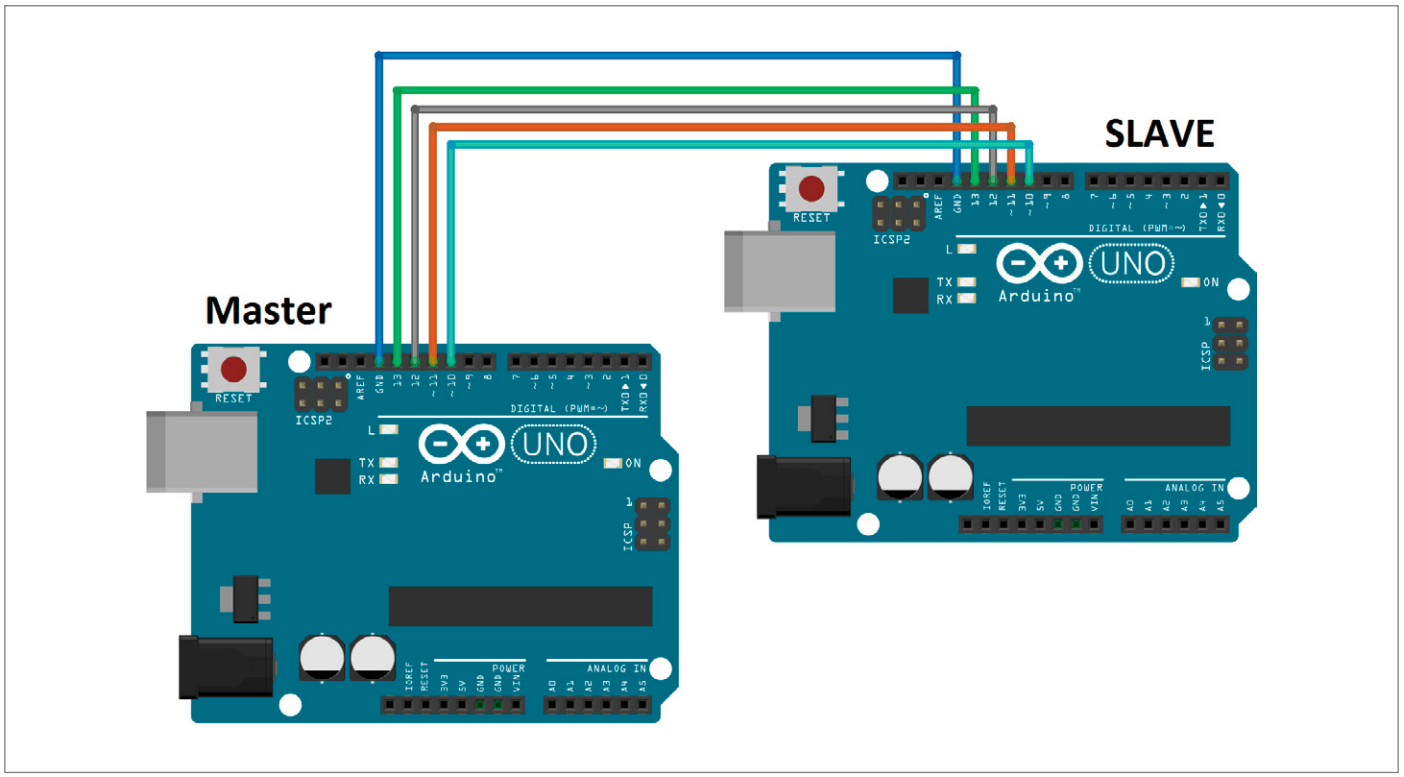

Bild 8: Zwei Arduinos kommunizieren via SPI.

Bild 7 zeigt die zugehörigen Anschlüsse an der SD-Karte. Wenn man ein fertiges Shield oder ein SD-Modul verwendet, muss man die Anschlüsse nicht gesondert betrachten.

Nach dem Aufstecken des Shields bzw. der Verbindung mit dem Modul sind alle erforderlichen Signale angeschlossen. Sollte ein anderer Arduino als der UNO zum Einsatz kommen, können die Pins abweichen.

Mit dem Beispielsketch (Datei > Beispiele > SD > CardInfo) der SD-Library kann man die Schaltung, den SD-Card-Reader und die SD-Karte selbst testen. Dabei muss nur angegeben werden, wie der SS- bzw. CS-Pin des Arduinos belegt ist (beim UNO z. B. D10).

SD-Karten sind eine gute Möglichkeit, die Speicherkapazität des Arduinos zu erweitern, insbesondere da sie billiger und größer sind als externe EEPROMs. Zusätzlich sind die Karten dank der SPI-Kommunikation einfach mit dem Arduino zu verbinden.

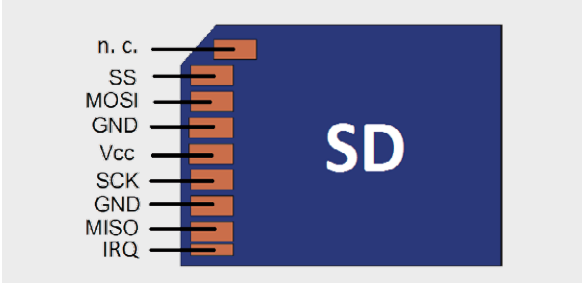

Bild 7: Auch SD-Karten werden über eine SPI-Schnittstelle angesprochen.

Mikrocontroller-Programmierung über SPI

Im Mikrocontrollerbereich, speziell auch für den Arduino, besitzt die SPI-Schnittstelle eine ganz besondere Bedeutung. Die Programmierung dieser Bauelemente bzw. Systeme erfolgt ebenfalls über die SPI-Schnittstelle. Ohne diese wäre also eine Programmierung gar nicht möglich. Solange ein Controller noch nicht über einen Bootloader verfügt, kann er mit einfachen Mitteln nur über den SPI-Port programmiert werden. Aus diesem Grunde verfügt auch praktisch jeder Arduino über die in Bild 3 dargestellte ICSP-Schnittstelle, obwohl ja alle Pins eigentlich bereits an anderer Stelle verfügbar wären. Die 6-Pin-Schnittstelle ist aber als standardisiertes Interface weit verbreitet und kann mit vielen sogenannten „Programmern" verbunden werden. Damit ist es auch möglich, einen Bootloader auf den Controller des Arduinos zu laden, falls dieser versehentlich gelöscht wurde.

Zudem ist der Arduino damit selbst als Programmer nutzbar. Hierfür muss lediglich ein entsprechender Sketch geladen werden. Damit ist es dann sogar möglich, weitere fabrikneue Controller mit einem Arduino-Bootloader zu versehen. Auf diese Weise kann sich der Arduino sozusagen selbst klonen. Hieraus ergibt sich ein erhebliches Einsparpotenzial, da ein einzelner Controllerchip natürlich wesentlich preisgünstiger ist als ein vollständiger Arduino. Details zu diesem Verfahren werden in einem späteren Artikel zu dieser Serie näher erläutert.

Von einem Arduino zum anderen: die SPI-Bridge

Abschließend soll noch gezeigt werden, wie der SPI-Bus zur Datenübertragung von einem Arduino zu einem anderen verwendet werden kann. Hierzu muss lediglich ein Board als Master und das andere als Slave konfiguriert werden.

Von der Hardwareseite her ist die Verbindung sehr einfach. Es sind nur wieder die vier bekannten Signale plus Masse erforderlich (siehe Bild 8):

- D10 (SS)
- D11 (MOSI)
- D12 (MISO)
- D13 (SCK)
- GND

In jedem Fall ist MOSI an einem Ende mit MOSI am anderen zu verbinden. Ein „Kreuzen der Leitungen" also MOSI <-> MISO ist nicht erforderlich. Diese Festlegung der Pin-Funktionen erfolgt in der Software. Dort wird ein Ende als MOSI (Master-Ende), also als Ausgang, und das andere Ende als Slave, d. h. als Eingang, definiert.

Damit die beiden Arduinos miteinander kommunizieren können, muss der folgende Sketch auf den Master geladen werden:

```
// master.ino

#include <SPI.h>

void setup (void)
{ digitalWrite(SS, HIGH);
  SPI on
  SPI.begin ();
  SPI.setClockDivider(SPI_CLOCK_DIV8);
}

void loop (void)
{ char c;
  digitalWrite(SS, LOW); // SS at D10
  for (const char * p = "ABC\n" ; c = *p; p++)
    SPI.transfer (c);
  digitalWrite(SS, HIGH);
  delay (1000);
}
```

Das Programm für den Slave sieht so aus:

```
// slave.ino

#include <SPI.h>

char buf [100];
volatile byte pos;
volatile bool process_it;

void setup (void)
{ Serial.begin (115200);
  SPCR |= bit (SPE);
  pinMode (MISO, OUTPUT);
  pos = 0;   // buffer empty
  process_it = false;
  SPI.attachInterrupt();
  Serial.println ("Slave active!");
}

ISR (SPI_STC_vect)
{ byte c = SPDR;
  if (pos < sizeof buf)
    { buf [pos++] = c;
      if (c == '\n')
      process_it = true;
    }
}

void loop (void)
{ if (process_it)
  { buf [pos] = 0;
    Serial.println (buf);
    pos = 0;
    process_it = false;
    }
}
```

Nach dem Laden der beiden Sketche sendet der Master den String „ABC" an den Slave. Der Empfang kann dort über die serielle Schnittstelle überprüft werden.

Ausblick

Nachdem in diesem Beitrag die Grundlagen und Anwendungen des SPI-Systems ausführlich dargelegt wurden, sollen im nächsten Beitrag die bislang noch nicht angesprochenen Interface-Systeme im Vordergrund stehen.

Neben dem „One-Wire"-Bus wird dann auch auf die etwas weniger bekannten Systeme, die häufig zur Ansteuerung von Temperatur oder Feuchtigkeitssensoren verwendet werden, eingegangen.

Ein Vergleich der wichtigsten Bussysteme wird das Thema dann abschließen.

Der Anwender sollte dann in der Lage sein, stets den für den konkreten Anwendungsfall am besten geeigneten Bus auszuwählen. ELV

Weitere Infos:

- Grundlagen zur elektronischen Schaltungstechnik finden sich in der E-Book-Reihe „Elektronik!" (www.amazon.de/dp/B000XNCB02)
- FRANZIS AVR-Mikrocontroller in C programmieren, Best.-Nr. CP-09 73 52
- Elektor-Praxiskurs AVR-XMEGA-Mikrocontroller, Best.-Nr. CP-12 07 62
- FRANZIS Arduino-Projects-Lernpaket, Best.-Nr. CP-11 51 22
- FRANZIS Physical Computing, Best.-Nr. CP-12 21 81
- FRANZIS Lernpaket Motoren & Sensoren mit Arduino, Best.-Nr. CP-12 74 74

Preisstellung Juli 2017 – aktuelle Preise im ELV Shop

Empfohlenes Produkt	Best.-Nr.	Preis
Arduino UNO	CP-10 29 70	€ 27,95

Alle Arduino-Produkte wie Mikrocontroller-Platinen, Shields, Fachbücher und Zubehör finden Sie unter: www.arduino.elv.de

Download-Paket zum Artikel:

Die Sketche und Beispieldateien zu diesem Artikel können heruntergeladen werden unter www.elv.de: Webcode #10113