

Serielle Kommunikation mit dem Arduino

Teil 1: Das Serial Peripheral Interface (SPI)

Axel

Attraktor e.V.

4. Juni 2012

Serielle Kommunikation mit dem Arduino: Teil 1

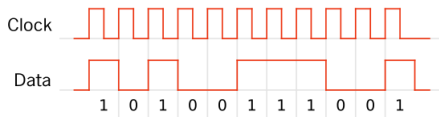
- 1 Überblick über serielle Kommunikation
- 2 SPI: Protokoll
- 3 SPI: Anbindung von ICs
- 4 Arduino-Bibliothek vs. Bit-Banging

Überblick über serielle Kommunikation

- Seriell: Bits werden einzeln nacheinander übertragen, codiert als Spannungsgröße (“strom an / strom aus”)



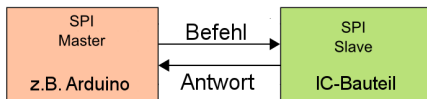
- 1 Leitung pro Richtung und gemeinsame Erde
- Oft wird dabei ein Takt benutzt der festlegt, wann die Datenleitungen interpretiert werden sollen:



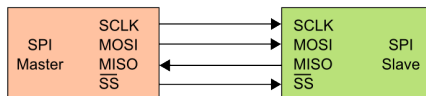
Jeder **Taktzyklus** besteht aus zwei **Taktflanken** (erste/zweite, vordere/hintere).

Der Serial Peripheral Interface Bus (SPI)

- SPI ist ein sehr einfaches, serielles Protokoll, welches für die Kommunikation zwischen Bauteilen oder Leiterplatten gedacht ist.
- Der **Master** steuert die Kommunikation, der **Slave** führt Befehle aus.
- Beliebige Kommunikation ist möglich, aber in fast allen Fällen werden einfache Befehls-Sprachen (Byte-Befehlscodes) verwendet.
 - ▶ DAC: "Schalte Ausgang 1 auf 2.43 Volt"
 - ▶ LCD: "Setze Bildschirmpunkt (32,24) auf Rot"
 - ▶ Speicher: "Lies Wert aus Speicherstelle 52452 aus".
- Viele Hersteller unterstützen es. Meist in leicht **unterschiedlichen Varianten** bzw. mit **unterschiedlichen Begriffen**.



SPI: Verbindung mit dem Bauteil



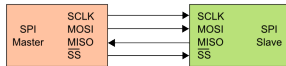
Name der Signale / Leitungen:

- Taktleitung / Clock: CLK, SCLK, SCK
- Daten vom Master zum Slave (Schreiboperation): MOSI, SIMO,
- Daten vom Slave zum Master (Leseoperation): MISO, SOMI
- Slave-Ansprache/Auswahl: SS (Slave Select), CS (Chip-Select)

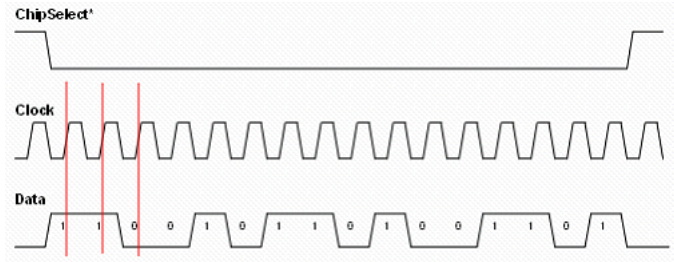
Alternativ aus der Sicht des Bauteils:

- Dateneingang eines Bauteils: SDI, DI, SI, IN, ...
- Datenausgang eines Bauteils: SDO, DO, SO, OUT, ...

SPI: Ablauf der Kommunikation



- 1 Slave-Select ("Active-Low") signalisiert dem Chip den Beginn eines Byte/Befehl
- 2 Chip interpretiert Datenleitung an Taktflanken



So weit, so einfach....

SPI: Clock Polarity und Clock Phase

... aber nun kommen **Clock Polarity** und **Clock Phase**....

Clock Polarity (CPOL):

- **CPOL=0**: Ruhewert der Taktleitung ist 0 (GND)
- **CPOL=1**: Ruhewert der Taktleitung ist 1 ((häufig) V_{CC})

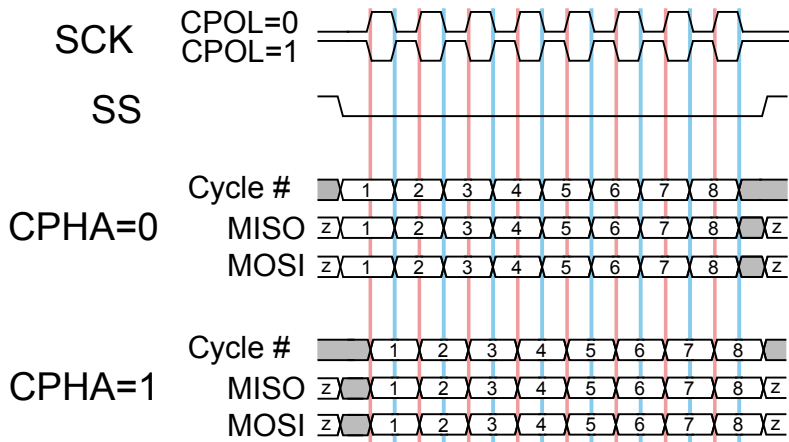
Clock Phase (CPHA):

- **CPHA=0**: Datenleitung wird an der ersten Flanke interpretiert und an der zweiten neu gesetzt.
- **CPHA=1**: Datenleitung wird an der ersten Flanke gesetzt und an der zweiten interpretiert.

Es ergeben sich 4 unterschiedliche SPI-Modi:

- Mode 0: CPOL=0, CPHA=0 / (0,0)
- Mode 1: CPOL=0, CPHA=1 / (0,1)
- Mode 2: CPOL=1, CPHA=0 / (1,0)
- Mode 3: CPOL=1, CPHA=1 / (1,1)

SPI: Clock Polarity und Clock Phase (II)



SPI: Senden als Master mit dem Arduino

Die SPI-Bibliothek des Arduino macht Kommunikation als SPI-Master einfach:

- `SPI.begin()`
startet die SPI-Bibliothek.
- `SPI.setBitOrder()`
setzt die Bit-Reihenfolge.
- `SPI.setClockDivider()`
stellt die Geschwindigkeit ein.
- `SPI.setDataMode()`
setzt den SPI-Modus.
- `SPI.transfer()`
überträgt ein Byte Daten.


Man stellt die richtigen Parameter mit den ersten Funktionen ein und kann danach mit `SPI.transfer()` Bytes übertragen.

Was muss nun wie eingestellt werden?

SPI: Kommunikation mit ICs

Ein Blick ins Datenblatt des Microchip MCP 4901-E/P 8-Bit Digital-Analog-Converter (DAC):

- Was muss wo angeschlossen werden?
- Welcher SPI-Modus (CPOL/CPHA) muss benutzt werden?
- Wie sieht das Befehls-Format aus?



MICROCHIP MCP4901/4911/4921

8/10/12-Bit Voltage Output Digital-to-Analog Converter with SPI Interface

Features

- MCP4901: 8-Bit Voltage Output DAC
- MCP4911: 10-Bit Voltage Output DAC
- MCP4921: 12-Bit Voltage Output DAC
- Rail-to-Rail Output
- SPI Interface with 20 MHz Clock Support
- Simultaneous Latching of the DAC Output with $\overline{\text{LDAC}}$ Pin
- Fast Settling Time of 4.5 μs
- Selectable Unity or 2x Gain Output
- External Voltage Reference Input
- External Multiplier Mode
- 2.7V to 5.5V Single-Supply Operation
- Extended Temperature Range: -40°C to $+125^{\circ}\text{C}$

Applications

- Set Point or Offset Trimming
- Precision Selectable Voltage Reference
- Motor Control Feedback Loop

Description

The MCP4901/4911/4921 devices are single channel 8-bit, 10-bit and 12-bit buffered voltage output Digital-to-Analog Converters (DACs), respectively. The devices operate from a single 2.7V to 5.5V supply with an SPI compatible Serial Peripheral Interface. The user can configure the full-scale range of the device to be V_{REF} or $2 \cdot V_{\text{REF}}$ by setting the gain selection option bit (gain of 1 of 2).

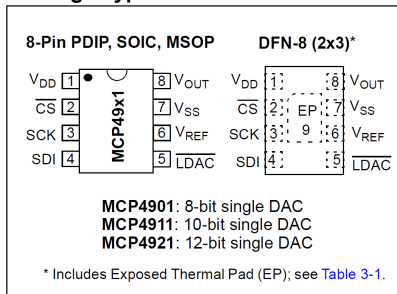
The user can shut down the device by setting the Configuration Register bit. In Shutdown mode, most of the internal circuits are turned off for power savings, and the output amplifier is configured to present a known high resistance output load (500 k Ω , typical).

The devices include double-buffered registers, allowing synchronous updates of the DAC output using the $\overline{\text{LDAC}}$ pin. These devices also incorporate a Power-on Reset (POR) circuit to ensure reliable power-up.

The devices utilize a resistive string architecture, with its inherent advantages of low Differential Non-Linearity (DNL) and Integral Non-Linearity (INL).

IC-Datenblatt (II): Chip-Anschlüsse

Package Types



Die Zeichnung zeigt, wie der Chip mit dem Arduino verbunden werden muss:

- Daten Master → Slave (MOSI): IC-Pin 4 (SDI) an Arduino Pin 11
- Takt (SCKL): IC-Pin 3 (SCK) an Arduino Pin 13
- Slave Select (SS): IC-Pin 2 (\overline{CS}) and Arduino Pin 10

MCP4901/4911/4921

5.0 SERIAL INTERFACE

5.1 Overview

The MCP4901/4911/4921 devices are designed to interface directly with the Serial Peripheral Interface (SPI) port, which is available on many microcontrollers and supports Mode 0,0 and Mode 1,1. Commands and data are sent to the device via the SDI pin, with data being clocked-in on the rising edge of SCK. The communications are unidirectional, thus the data cannot be read out of the MCP4901/4911/4921. The $\overline{\text{CS}}$ pin must be held low for the duration of a write command. The write command consists of 16 bits and is used to configure the DAC's control and data latches. Register 5-1 through Register 5-3 detail the input register that is used to configure and load the DAC register for each device. [Figure 5-1](#) through [Figure 5-3](#) show the write command for each device.

Refer to [Figure 1-1](#) and the SPI Timing Specifications Table for detailed input and output timing specifications for both Mode 0,0 and Mode 1,1 operation.

5.2 Write Command

The write command is initiated by driving the $\overline{\text{CS}}$ pin low, followed by clocking the four Configuration bits and the 12 data bits into the SDI pin on the rising edge of SCK. The $\overline{\text{CS}}$ pin is then raised, causing the data to be latched into the DAC's input register.

The MCP4901/4911/4921 utilizes a double-buffered latch structure to allow the analog output to be synchronized with the LDAC pin, if desired.

By bringing the LDAC pin down to a low state, the content stored in the DAC's input register is transferred into the DAC's output register (V_{OUT}), and V_{OUT} is updated.

All writes to the MCP4901/4911/4921 devices are 16-bit words. Any clocks past the 16th clock will be ignored. The Most Significant 4 bits are Configuration bits. The remaining 12 bits are data bits. No data can be transferred into the device with $\overline{\text{CS}}$ high. This transfer will only occur if 16 clocks have been transferred into the device. If the rising edge of $\overline{\text{CS}}$ occurs prior to that, shifting of data into the input register will be aborted.

- Es werden die SPI-Modes (0,0) und (1,1) unterstützt.
- Befehle haben immer 16 Bit.
- Man kann nur Befehle senden, keine Daten auslesen.

IC-Datenblatt (IV): Befehlsformat

REGISTER 5-3: WRITE COMMAND REGISTER FOR MCP4901 (8-BIT DAC)

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
0	BUF	GA	SHDN	D7	D6	D5	D4	D3	D2	D1	D0	x	x	x	x
bit 15								bit 0							

Where:

bit 15 0 = Write to DAC register
 1 = Ignore this command

bit 14 **BUF:** V_{REF} Input Buffer Control bit
 1 = Buffered
 0 = Unbuffered

bit 13 **GA:** Output Gain Selection bit
 1 = $1\times (V_{OUT} = V_{REF} * D/4096)$
 0 = $2\times (V_{OUT} = 2 * V_{REF} * D/4096)$

bit 12 **SHDN:** Output Shutdown Control bit
 1 = Active mode operation. V_{OUT} is available.
 0 = Shutdown the device. Analog output is not available. V_{OUT} pin is connected to 500 k Ω (typical).

bit 11-0 **D11:D0:** DAC Input Data bits. Bit x is ignored.

IC-Datenblatt (V): Befehle per SPI senden

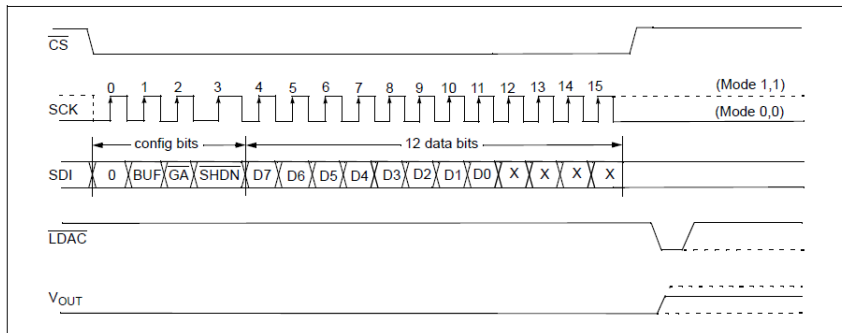


FIGURE 5-3: Write Command for MCP4901(8-bit DAC). Note: X are don't care bits.

- Das höchstwertige bit (Most significant bit / MSB) wird zuerst gesendet.

Senden als Master: Setup der Arduino-Bibliothek

```
#include <SPI.h>
byte pinSS = 10;

void setup() {
    SPI.begin();
    // Bit-Reihenfolge (LSBFIRST oder MSBFIRST)
    SPI.setBitOrder(MSBFIRST);

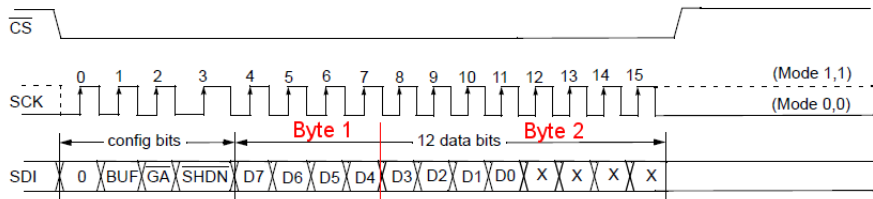
    // Geschwindigkeit auf halbe Arduino Taktrate
    SPI.setClockDivider(SPI_CLOCK_DIV2)

    // SPI-Mode (1,1): MODE3
    SPI.setDataMode(SPI_MODE3);

    // Slave Select Pin auf OUTPUT setzen
    pinMode(pinSS, OUTPUT);
}
```


Senden als Master (II): Bytes senden

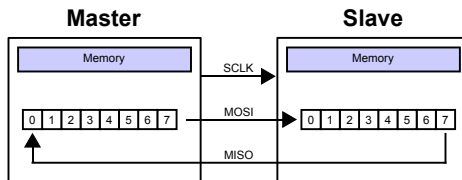
```
const int cmdFlags = 0x30;  
  
void sendToDAC(byte value) {  
    digitalWrite(pinSS, LOW);  
    SPI.transfer(cmdFlags | (value >> 4));  
    SPI.transfer(value << 4);  
    digitalWrite(pinSS, HIGH);  
}
```



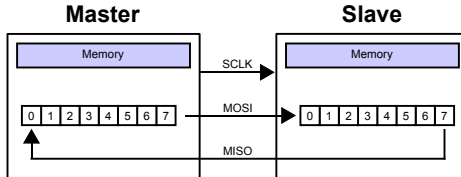
SPI: Auslesen von Daten aus dem SPI-Slave

- Aktivität geht bei SPI immer vom Master aus, der Slave kann *nicht* von sich aus Daten senden.
- Der SPI-Master sendet einen Lese-Befehl und der Slave sendet anschließend die Daten zurück.'
- Der Takt wird immer vom Master generiert!

Der innere Aufbau der SPI-Hardware: Ring-verbundene Schieberegister:



Jeder `SPI.transfer()`-Aufruf ist gleichzeitig Sende- und Empfangsbefehl.



```
byte spi_read(byte read_command) {  
    byte miso;  
    digitalWrite(pinSS, LOW);  
    SPI.transfer(read_command);  
    miso_data = SPI.transfer(0);  
    digitalWrite(pinSS, HIGH);  
    return miso_data;  
}
```

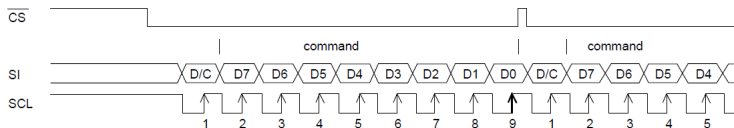
Nach dem senden des Lese-Befehls wird ein Leer-Befehl übertragen um die Daten einzulesen, die der Chip im Anschluss sendet.

SPI: Verarbeitung anderer Befehlslängen

Aus dem Datenblatt eines LCD-Controllers:

S1D15G10D08B000

(2) 9-bit serial interface



Die SPI-Bibliothek des Arduino bzw. die Hardware des ATmega unterstützen nur Transfer in 8-Bit-Bytes. Befehlslängen müssen also Vielfache von 8 sein.

Wie kann dieser Controller angesteuert werden?

SPI: Bit-Banging des SPI-Protokolls

“Bit-Banging” ist der (inoffizielle) Name der Technik, ein Protokoll direkt durch Ansteuerung von I/O-Pins mit den Funktionen

- `digitalRead()`
- `digitalWrite()`

zu implementieren, also nicht über die Arduino-Bibliothek.

Dies ist für alle Protokolle notwendig, die nicht von einer Arduino-Bibliothek unterstützt werden. Da SPI ein sehr einfaches Protokoll ist, ist auch das entsprechende Bit-Banging wenig aufwendig.

SPI-Bitbanging: Setup

```
const byte pinSS    = 2;  
const byte pinSCKL  = 3;  
const byte pinMOSI  = 4;  
  
void setup() {  
  
    // Alle pins auf OUTPUT setzen  
    pinMode(pinSS, OUTPUT);  
    pinMode(pinSCKL, OUTPUT);  
    pinMode(pinMOSI, OUTPUT);  
}
```

SPI-Bitbanging: Senden eines 9-Bit-Befehls

```
void spi_send_to_LCD(int value) {  
    byte c;  
    digitalWrite(pinSS, LOW);  
    for(c=0; c<= 8; c++) { // 9 bits!  
        if(value & (1 << 8)) { // MSB des 9-Bit Wertes  
            digitalWrite(pinMOSI, HIGH);  
        } else {  
            digitalWrite(pinMOSI, LOW);  
        }  
        // Einen Takt signalisieren  
        digitalWrite(pinSCKL, LOW);  
        digitalWrite(pinSCKL, HIGH);  
        // Naechstes Sende-Bit vorbereiten  
        value = (value << 1);  
    }  
    digitalWrite(pinSS, HIGH);  
}
```

SPI-Bitbanging: Vergleich zur Bibliothek

- Vorteile von Bitbanging
 - ▶ Volle Flexibilität bei Protokoll-Optionen
 - ▶ Ermöglicht Implementation nicht-unterstützter Protokolle
- Nachteile von Bitbanging
 - ▶ Geringere, erreichbare Höchstgeschwindigkeit
 - ▶ Kein Transfer im Hintergrund. CPU ist voll gebunden
 - ▶ Absolutes Timing schwierig
 - ▶ Genau gleichzeitiges Ändern von Pegeln erfordert zusätzliche Hardware
 - ▶ Code für komplexere Protokolle wird schnell unübersichtlich

Das war

Serielle Kommunikation

Teil 1: Serial Peripheral Interface

Fragen? :)

Es folgen noch:

- I2C
- UART und RS232