C4 Die SPI Schnittstelle

Einführung

Motorola entwickelte die **synchrone SPI-Master-Slave Schnittstelle**, (***Serial Periphal Interface***) für die Kommunikation zwischen Mikrocontrollern. Ein ähnliches Bus-System existiert von National Semiconductor und nennt sich **Microwire**. Die Schnittstelle arbeitet synchron in Vollduplex mit hoher Taktgeschwindigkeit (bis zu mehreren 10 MHz). Die SPI-Schnittstelle wird bei der AVR-Familie zur *In-System*-Programmierung genutzt und meist hardwaremäßig unterstützt.

Im Gegensatz zur asynchronen seriellen EIA-232 Schnittstelle werden keine Start-, Stopp- oder Paritätsbits verwendet. Auch wird keine Adresse wie bei der synchronen I^2C-Schnittstelle benötigt.

Vorteile Schnell, da Vollduplex. Die Taktrate kann bis zur Hälfte des Systemtakts des Controllers betragen. Es werden keine Steuerbits benötigt. Es können mehrere Master abwechselnd am Bus arbeiten.

Nachteile Die überbrückbare Distanz ist gering. Der Verdrahtungsaufwand ist meist höher als bei I^2C (3 bis 5 Leitungen).

Verwendung: Datenaustausch zwischen Mikrocontrollern, aber auch mit anderen ICs (z.B. EEPROM) oder Sensoren welche über diese Schnittstelle verfügen. Zusätzliche Ein/Ausgabeports mittels TTL-Schieberegistern (Wandlung seriell-parallel).

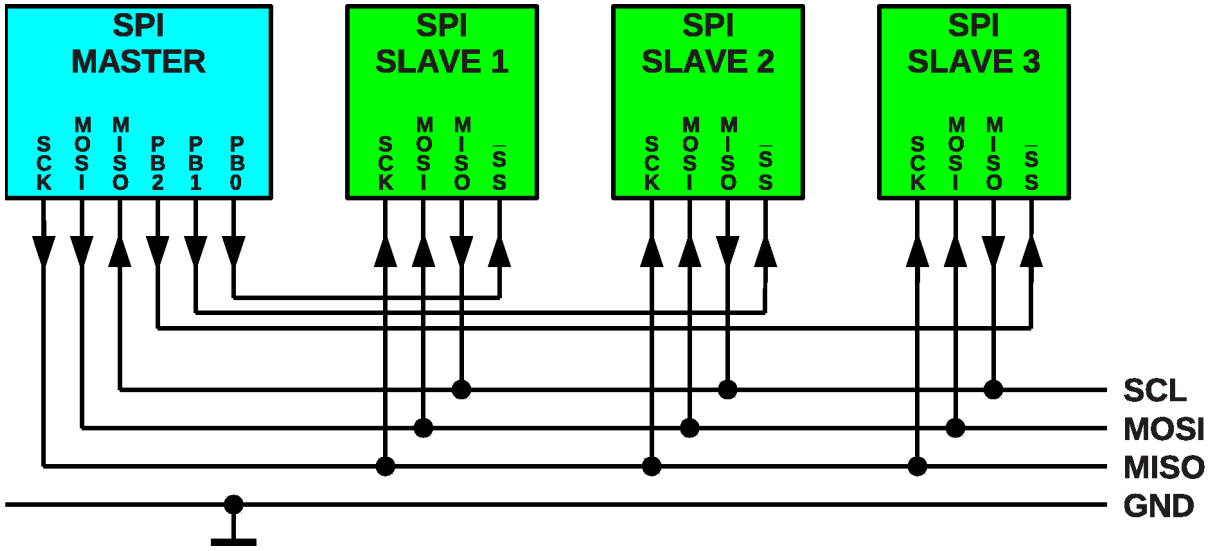
Die SPI Schnittstelle arbeitet mit 8-Bit Schieberegistern. Es kann frei gewählt werden, mit welcher Geschwindigkeit (mit Vorteiler aus dem Systemtakt abgeleitet) gearbeitet wird und ob MSB oder LSB zuerst gesendet werden.

Aufbau der SPI-Schnittstelle

Verdrahtung und Schieberichtung

Im folgenden Bild sind ein Master und drei Slaves eingezeichnet. Beliebige Ausgänge (hier PB0-PB2) dienen dem Master, zum Auswählen des anzusteuernden Bausteins (***/SS Slave Select***). Die "Slave Select"-Eingänge sind dabei **Low-Aktiv!** Bemerkt der Slave, dass seine SS-Leitung auf Low gezogen wurde (fallende Flanke), so beginnt für ihn die Übertragung. Diese Leitung dient also auch der Synchronisation der Übertragung zwischen Master und Slave.

Die SPI-Schnittstelle arbeitet synchron, d.h. mit einer gemeinsamen Taktleitung. Auf der ***Serial Clock*** Leitung (**SCK**) legt der Master das Taktsignal an, solange wie SS Low ist.



Dazu muss allerdings mit einem Schreiben in das SPI-Datenregister die Kommunikation gestartet werden (siehe später).

MOSI bedeutet ***Master Out Slave In*** und wird 1:1 verbunden. Für den Master ist diese Leitung ein Ausgang (muss initialisiert werden) und beim Slave automatisch ein Eingang. Der Master sendet über diese Leitung. **MISO** bedeutet ***Master In Slave Out*** und wird auch 1:1 verbunden. MISO ist für den Master automatisch ein Eingang. Er empfängt Daten vom Slave über diese Leitung. Vollduplex ist also mit der SPI Schnittstelle möglich. Beim Slave muss dieses Pin als Ausgang initialisiert werden.

Pin	Master	Slave
MOSI	als Ausgang zu initialisieren	automatisch Eingang
MISO	automatisch Eingang	als Ausgang zu initialisieren
SCK	als Ausgang zu initialisieren	automatisch Eingang
/SS	als Ausgang zu initialisieren	automatisch Eingang

Es soll hier nicht auf den möglichen Multimasterbetrieb eingegangen werden. Dass /SS Pin des Masters muss für den Single Master-Betrieb als Ausgang gesetzt werden!

Je nach angeschlossenen Slave muss die Schnittstelle flexibel konfigurierbar sein. Mit Hilfe des **DORD**-Bit im Kontrollregister kann die Schieberichtung festgelegt werden. Es besteht die Möglichkeit das höchstwertige Bit zuerst (**DORD** = 0) rauszuschieben, oder mit dem niederwertigsten Bit zu beginnen(**DORD** = 1).

Mit zwei weiteren Bits lässt sich auch die Taktleitung und der Zeitpunkt der Übernahme von Daten beeinflussen:

Die SPI-Modi und die Geschwindigkeit

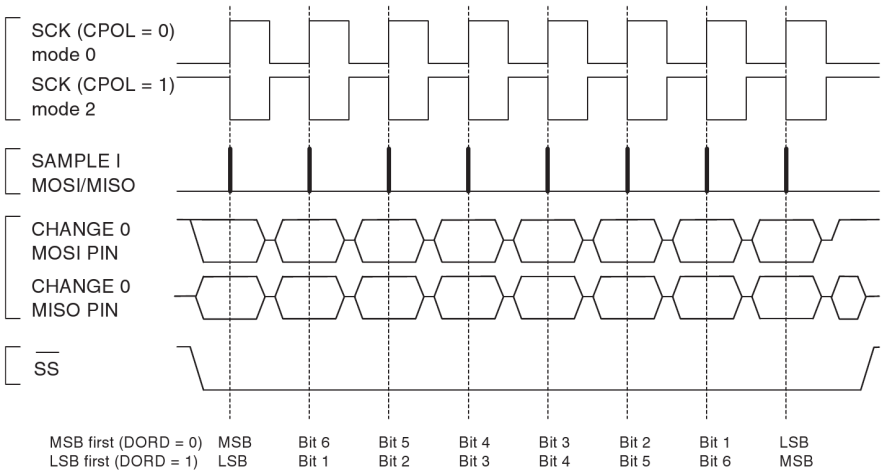
Mit den beiden Bits **CPOL** und **CPHA** kann einer von vier SPI-Modi ausgewählt werden. **CPOL** bestimmt dabei die Polarität der SCK-Leitung (0: Ruhezustand der Taktleitung Low, 1: Ruhezustand High) und **CPHA** die Phasenlage d.h bei welcher Flanke die Daten übernommen werden sollen (0: erste Flanke sofort, 1: zweite Flanke nach halber Taktzeit (180°)).

Zwischen dem Abtasten (sample) und der Ausgabe (Schieben) des Bits bleibt immer eine halbe Periode Zeit, damit die Signale sich stabilisieren können.

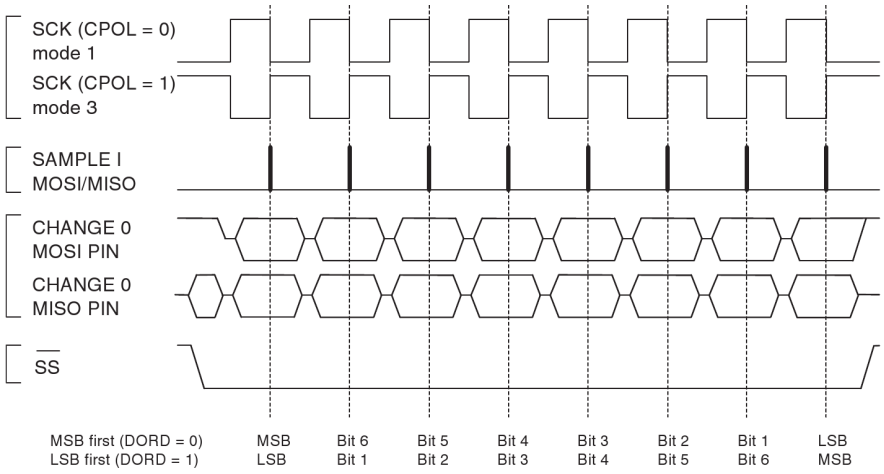
SPI-Modus	CPOL	CPHA	erste Flanke	zweite Flanke
0	0	0	Bit abtasten (steigende Flanke)	Bit ausgeben (fallende Flanke)
1	0	1	Bit ausgeben (steigende Flanke)	Bit abtasten (fallende Flanke)
2	1	0	Bit abtasten (fallende Flanke)	Bit ausgeben (steigende Flanke)
3	1	1	Bit ausgeben (fallende Flanke)	Bit abtasten (steigende Flanke)

Im Modus 0 legt der Slave seine Daten schon beim Runterziehen von SS an MISO an. Der Master kann sie dann beim ersten Flankenwechsel übernehmen. Hier ist auf das Timing zu achten (ausreichen Zeit zwischen beiden Aktionen damit kein Bit verloren geht).

CPHA = 0



CPHA = 1



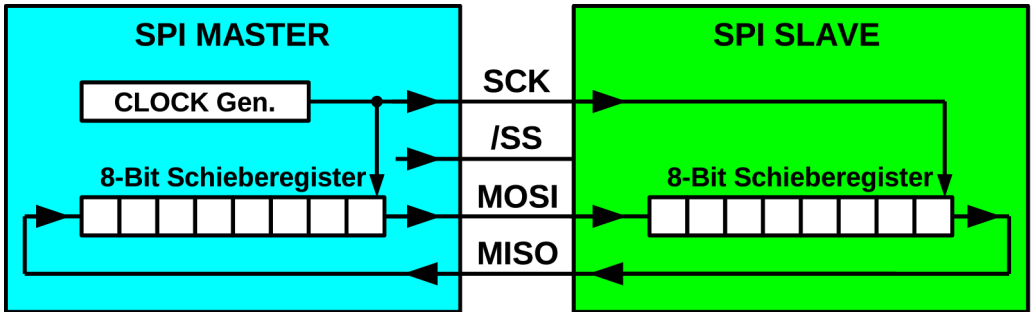
Quelle: Datenblatt ATmega32A

Die Frequenz des SPI-Taktsignals ist an sich beliebig und wird durch einen Teilungsfaktor festgelegt. Die Taktfrequenz des Controllers (Systemtakt) wird dabei durch 2, 4, 8, 16, 32, 64 oder 128 geteilt. Die maximale Geschwindigkeit des Masters liegt mit 16 MHz Quarz also bei 8 MHz. Die Geschwindigkeit des Slave soll allerdings nicht höher als Systemtakt/4 sein!

Funktionsweise der SPI-Schnittstelle

Der Master legt den Schiebetakt auf die SCK-Leitung, nachdem er ein Byte im Schieberegister abgelegt hat. Mit jeder Periode des Taktsignals wird 1 Bit vom Master-Schieberegister ins Slave-Schieberegister und gleichzeitig vom Slave-Schieberegister ins Master-Schieberegister geschoben. Nach 8 Taktzyklen ist die Übertragung abgeschlossen. Die beiden 8-Bit Schieberegister können als großes 16-Bit-Schieberegister betrachtet werden!!

Bei jeder Datenübertragung wird immer gleichzeitig ein Byte gesendet und empfangen!



Will der Master nur Senden, so ignoriert er die vom Slave ankommenden Bytes, d.h er liest sie nicht ein. Will er nur Empfangen, so ignoriert der Slave die nichtssagenden ankommenden Bytes vom Master.

Sowohl beim Master als auch beim Slave wird das Ende der Übertragung eines Byte, nach 8 Taktzyklen durch das SPI-Interrupt-Flag **SPIF** im SPI-Statusregister angezeigt. Dieses kann mittels Polling abgefragt werden oder einen Interrupt auslösen.

Daten können im SPI-Datenregister **SPDR** abgelegt oder empfangen werden, nachdem **SPIF** gesetzt wurde.

Schreibt man in das Datenregister **SPDR**, so werden die Daten im Sendepuffer abgelegt, bevor sie ins Schieberegister wandern. Da der Sendepuffer nur ein Byte groß ist, sind weitere Schreibzugriffe ins Datenregister während einer Übertragung gesperrt. Wird dies trotzdem versucht, so wird die Übertragung nicht gestört, sondern die Schreib-Kollision wird mit dem Flag **WCOL** (*Write COLlision*) im Statusregister **SPSR**. Bei Bedarf kann dies kontrolliert werden. Um Kollisionen zu vermeiden soll immer unmittelbar nach einer Übertragung (SPIF = 1) ins Datenregister geschrieben werden.

Der Lesebuffer beträgt zwei Bytes. Liest man Daten aus dem Datenregister **SPDR**, so kann dies auch noch während der Übertragung des folgenden Bytes geschehen. Das Lesen muss allerdings abgeschlossen sein bevor die Übertragung beendet wird.

Die Initialisierung der SPI-Schnittstelle

Damit eine Kommunikation per SPI möglich ist, müssen Sender und Empfänger richtig initialisiert werden.

4. Die Schnittstelle muss eingeschaltet werden, und es muss festgelegt werden ob der Controller als Master oder Slave arbeitet (**SPE**, **MSTR**).

5. Beim Master müssen **MOSI**, **SCK** und **/SS** als Ausgang initialisiert werden, beim Slave nur **MISO** als Ausgang.

6. Die Schieberichtung und der verwendete SPI-Modus müssen festgelegt werden (**DORD**, **CPOL**, **CPHA**).

7. Die Geschwindigkeit muss festgelegt werden (**SPR0**, **SPR1**[17])

8. Wird mit Interrupts statt Polling gearbeitet, so muss neben der Initialisierung des Interrupt-Vektors, dem globalen Freischalten von Interrupts auch der SPI-Interrupt freigeschaltet werden (**SPIE**)

Die SPI Schnittstelle benötigt nur drei SF-Register.

17 Nur im Fall, dass die höchst mögliche Geschwindigkeit nötig ist oder der Teilungsfaktor 8 bzw. 32 benötigt wird, muss zusätzlich das Bit zur Verdopplung der Geschwindigkeit im SPI-Statusregister gesetzt werden (SPI2X in SPSR)

- **SPCR** Die gesamte Initialisierung kann im **SPI-Kontrollregister** vorgenommen werden.
- **SPSR** Das **SPI-Statusregister** dient hauptsächlich der Abfrage des SPI Interrupt-Flags **SPIF** beim Polling.
- **SPDR** Daten werden über das **SPI-Datenregister** in den Sendepuffer geschrieben bzw. aus dem Empfangspuffer gelesen.

Die SF-Register der SPI-Schnittstelle

Das SPI Control Register SPCR

Das **SPI Control Register** befindet sich auf der SRAM-Adresse **0x002D** (SF-Register-Adresse **0x0D**) und wird mit der Abkürzung "**SPCR**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können verwendet werden.

SPCR = SPI Control Register

Bit	7	6	5	4	3	2	1	0
SPCR 0x0D	**SPIE**	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPIE ***SPI Interrupt Enable***

- **0** kein SPI-Interrupt erlaubt.
- **1** In dem Moment, wo die Übertragung aller 8 Bit beendet ist wird das SPI Interrupt-Flag **SPIF** im Statusregister **SPSR** gesetzt. Das Setzen des **SPIE**-Flag **ermöglicht das Auslösen eines Interrupts** sobald **SPIF** gesetzt wurde. Interrupts müssen dazu global frei gegeben sein (**I** = 1 im **SREG** mit "**sei**"). **SPIF** wird hardwaremäßig gelöscht wenn der Interrupt abgearbeitet wird.

SPE ***SPI Enable***

- **0** schaltet SPI aus.
- **1** schaltet SPI ein. Muss gesetzt werden, damit SPI-Operationen durchgeführt werden.

DORD ***Data ORDer***

- **0** das höchstwertigste Bit **MSB** wird als Erstes gesendet.
- **1** das niederwertigste Bit **LSB** wird als Erstes gesendet.

MSTR ***Master/Slave Select***

- **0** Controller ist Slave.
- **1** Controller ist Master. Das Bit muss vor oder gleichzeitig mit dem **SPE** Bit gesetzt werden!

CPOL ***Clock Polarity***

- **0** Ruhezustand der Taktleitung ist Low.
- **1** Ruhezustand der Taktleitung ist High.

CPHA ***Clock PHAse***

0 Daten werden sofort bei der ersten Flanke übernommen.
1 Daten werden bei der zweiten Flanke nach der halben Taktzeit (180°) übernommen.

Mit **CPOL** und **CPHA** wird der SPI-Modus festgelegt:

CPOL CPHA	SPI Modus:
00	0
01	1
10	2
11	3

SPRn ***SPI Clock Rate Select*** ***SPR2X, SPR1, SPR0***

Mit drei Bit wird die Taktgeschwindigkeit (Frequenz) des Busses **SCK** festgelegt. Dies betrifft nur den Master. Das Bit **SPR2X** befindet sich im SPI Statusregister **SPSR** (Bit 0).

SPR2X SPR1 SPR0 2^2 2^1 2^0	SCK Frequenz:
000	**Systemtakt / 4**
001	**Systemtakt / 16**
010	**Systemtakt / 64**
011	**Systemtakt / 128**
100	Systemtakt / 2
101	Systemtakt / 8
110	Systemtakt / 32
111	Systemtakt / 64

Weitere Informationen findet man im Datenblatt S 137.

Das SPI Status Register SPSR

Das **SPI Status Register** befindet sich auf der SRAM-Adresse **0x002E** (SF-Register-Adresse **0x0E**) und wird mit der Abkürzung "**SPSR**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können verwendet werden.

SPSR = SPI Status Register

Bit	7	6	5	4	3	2	1	0
SPSR 0x0E	**SPIF**	**WCOL**	**-**	**-**	**-**	**-**	**-**	**SPI2X**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R/W

SPIF ***SPI Interrupt Flag***

0 Übertragung noch nicht beendet.

1 In dem Moment, wo die Übertragung aller 8 Bit beendet ist wird das SPI Interrupt-Flag **SPIF** gesetzt. Das Setzen des **SPIE**-Flag in **SPCR** ermöglicht das Auslösen eines Interrupts. **SPIF** wird hardwaremäßig gelöscht wenn der Interrupt abgearbeitet wird. Wird **SPIF** im Polling-Betrieb abgefragt, so kann das Flag durch Lesen des Statusregisters **SPSR** bei gesetztem **SPIF** und anschließendem Lesen oder Schreiben des Datenregisters **SPDR** gelöscht werden.

WCOL ***Write COLlision Flag***

0 keine Schreibkollision aufgetreten.

1 Wird **SPDR** während der Datenübertragung beschrieben, so meldet das **WCOL**-Flag eine Schreibkollision. Das Flag kann (zusammen mit **SPIF**) durch Lesen des Statusregisters **SPSR** bei gesetztem **WCOL** und anschließendem Lesen oder Schreiben des Datenregisters **SPDR** gelöscht werden.

SPI2X ***Double SPI Speed Bit***

0 keine Verdopplung der Taktgeschwindigkeit des Busses.

1 Verdopplung der Taktgeschwindigkeit des Busses (siehe Tabelle **SPCR**).

Das SPI Data Register SPDR

Das **SPI Data Register** befindet sich auf der SRAM-Adresse **0x002F** (SF-Register-Adresse **0x0F**) und wird mit der Abkürzung "**SPDR**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können verwendet werden. Das Register ist les- und schreibbar.

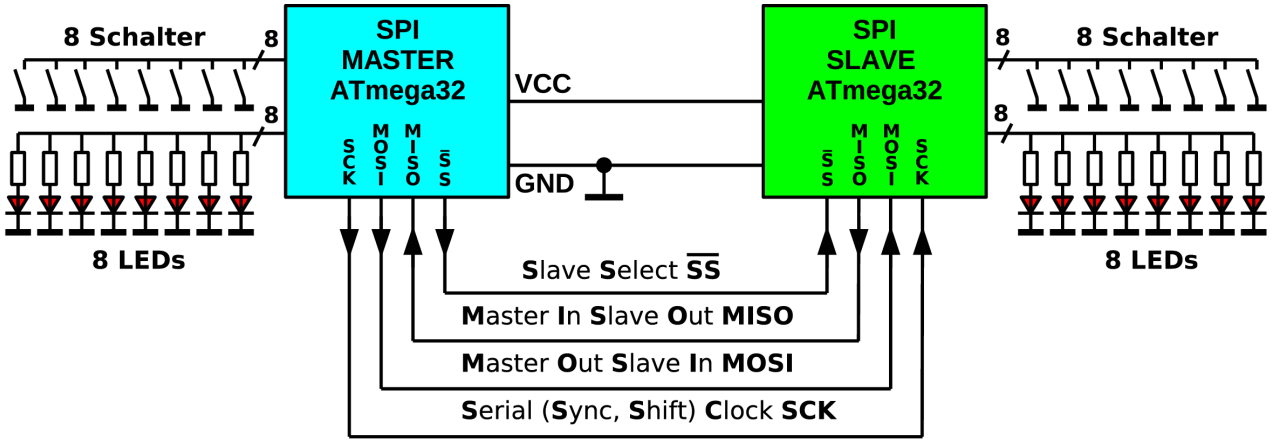
SPDR = SPI Data Register

Bit	7 (MSB)	6	5	4	3	2	1	0 (LSB)
SPDR 0x0F	**SPDR7**	**SPDR6**	**SPDR5**	**SPDR4**	**SPDR3**	**SPDR2**	**SPDR1**	**SPDR0**
Startwert	-	-	-	-	-	-	-	-
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Der Startwert ist undefiniert.

Ansteuerung eines Slaves mittels Polling

Mittels zweier ATmega32A soll eine SPI-Übertragung in beide Richtungen zwischen einem Master und einem Slave realisiert werden. Beide Controller besitzen 8 Schalter und 8 LEDs. Die Schalterstellung des Master soll an den LEDs des Slave sichtbar sein und umgekehrt.



Programm für den Master:

```
        ;Schalter und LEDs initialisieren
        clr     Tmp1            ;8 Schalter an PORTA
        out     DDRA,Tmp1
        ser     Tmp1
        out     PORTA,Tmp1      ;Pull-Ups
        out     DDRC,Tmp1       ;8 LEDs an PORTC

        ;SPI Master initialisieren
        ;SPI-Ausgaenge initialisieren (PB6 (MISO) automatisch Eingang)
        sbi     DDRB,4          ;PB4 Ausgang (/SS) muss beim Master Mode vor dem
                                ;Einschalten der Schnittstelle geschehen!
        sbi     DDRB,5          ;PB5 Ausgang (MOSI)
        sbi     DDRB,7          ;PB7 Ausgang (SCK)
        ;SPI einschalten, Master mit SCK = Systemtakt/128, MSB first, SPI Mode 1
        ldi     Tmp1,0x57       ;SPCR = 0b01010111
        out     SPCR,Tmp1       ;B7 SPIE=0 kein Interrupt
                                ;B6 SPE=1 SPI einschalten
                                ;B5 DORD=0 MSB zuerst
                                ;B4 MSTR=1 Master
                                ;B32 CPOL=0 CPHA=1 SPI Modus 1
                                ;B10 SPR1=1 SPR0=1 Systemtakt/128 (125khz)
        ;Slave aktivieren
        cbi     PORTB,4         ;Slave aktivieren

;-----------------------------------------------------------------------------
;       Hauptprogramm
;-----------------------------------------------------------------------------
MAIN:   in      Tmp1,PINA       ;Schalter einlesen und ueber SPI uebertragen
        out     SPDR,Tmp1       ;Starte die Uebertragung
W_SPIF: sbis    SPSR,SPIF       ;Warte bis Uebertragung fertig
        rjmp    W_SPIF
        in      Tmp1,SPDR       ;Empfange Byte vom Slave
        out     PORTC,Tmp1      ;und Ausgabe an LEDs
        rjmp    MAIN            ;Endlosschleife
```

Programm für den Slave:

```
        ;Schalter und LEDs initialisieren
        clr     Tmp1            ;8 Schalter an PORTA
        out     DDRA,Tmp1
        ser     Tmp1
        out     PORTA,Tmp1      ;Pull-Ups
        out     DDRC,Tmp1       ;8 LEDs an PORTC

        ;SPI Slave initialisieren
        ;SPI-Ausgang initialisieren
        ;(PB4(/SS), PB5(MOSI) und PB7(SCK) automatisch Eingang)
        sbi     DDRB,6          ;PB6 Ausgang (MISO)
        ;SPI einschalten, Slave mit SCK = Systemtakt/128, MSB first, SPI Mode 1
        ldi     r16,0x47        ;SPCR = 0b01000111
        out     SPCR,r16        ;B7 SPIE=0 kein Interrupt
                                ;B6 SPE=1 SPI einschalten
                                ;B5 DORD=0 MSB zuerst
                                ;B4 MSTR=0 Slave
                                ;B32 CPOL=0 CPHA=1 SPI Modus 1
                                ;B10 SPR1=0 SPR0=1 Systemtakt/128 (125khz)

;-----------------------------------------------------------------------------
;       Hauptprogramm
;-----------------------------------------------------------------------------
MAIN:   in      Tmp1,PINA       ;Schalter einlesen
W_SPIF: sbis    SPSR,SPIF       ;Warte bis Empfang fertig
        rjmp    W_SPIF
        out     SPDR,Tmp1       ;Daten ueber SPI uebertragen
        in      Tmp1,SPDR       ;Empfangene Daten einlesen
        out     PORTC,Tmp1      ;und auf LEDs ausgeben
        rjmp    MAIN            ;Endlosschleife
```

✎ **C400** Teste die Übertragung mit den beiden vorgegebenen Programme[18]. Schreibe die Programme so um, dass die Initialisierung und das Polling jeweils in einem Unterprogramm erfolgen.
Nenne die neuen Programme "**C400_SPI_master_polling.asm**" und "**C400_SPI_slave_polling.asm**".

Bemerkungen: Nach dem Programmieren ist ein Reset des Masters notwendig, damit der Slave richtig synchronisiert.
Bei zu hohen Frequenzen (16 MHz/4 = 4 MHz) und mit CHPA = 0 (Modus 0) funktionierte die Synchronisation nicht zuverlässig. Abhilfe schafft das konsequente Einsetzen der /SS Leitung um den Slave vor der Übertragung eines jeden Bytes zu synchronisieren.
Acht auf eine gemeinsame Masse zwischen beiden Schaltungen!

Ansteuerung eines Slaves mittels Interrupt

Die gleiche Aufgabe soll nun mit Interrupts erledigt werden.

Programm für den Master:

```
;-----------------------------------------------------------------------------
;       Sprungadressen fuer die Interrupts organisieren (ISR VECTORS)
;-----------------------------------------------------------------------------
        ;Vektortabelle (im Flash-Speicher)
.ORG    SPIaddr                 ;interner Vektor fuer SPI (alt.: .ORG 0x0018)
```

18 Download auf www.weigu.lu/a/asm

```
       rjmp    ISRSPI          ;Springe zur ISR von SPI Transfer Ready
```

...

```
        ;Schalter und LEDs initialisieren
        clr     Tmp1            ;8 Schalter an PORTA
        out     DDRA,Tmp1
        ser     Tmp1
        out     PORTA,Tmp1      ;Pull-Ups
        out     DDRC,Tmp1       ;8 LEDs an PORTC
        ;SPI Master initialisieren
        ;SPI-Ausgaenge initialisieren (PB6 (MOSI) automatisch Eingang)
        sbi     DDRB,4          ;PB4 Ausgang (/SS) muss beim Master Mode vor dem
                                ;Einschalten der Schnittstelle geschehen!
        sbi     DDRB,5          ;PB5 Ausgang (MOSI)
        sbi     DDRB,7          ;PB7 Ausgang (SCK)
        ;SPI einschalten, Master mit SCK = Systemtakt/128, MSB first, SPI Mode 1
        ldi     r16,0xD7        ;SPCR = 0b11010111
        out     SPCR,r16        ;B7 SPIE=1 Interrupt !!
                                ;B6 SPE=1 SPI einschalten
                                ;B5 DORD=0 MSB zuerst
                                ;B4 MSTR=1 Master
                                ;B32 CPOL=0 CPHA=1 SPI Modus 1
                                ;B10 SPR1=0 SPR0=1 Systemtakt/128 (125 kHz)
        ;Slave aktivieren
        cbi     PORTB,4         ;Slave aktivieren

;-----------------------------------------------------------------------------
;       Hauptprogramm
;-----------------------------------------------------------------------------
MAIN:   sei                     ;Interrupts Global erlauben
        clr     Tmp1
        out     SPDR,Tmp1       ;Uebertragung starten
END:    rjmp    END             ;Endlosschleife

;-----------------------------------------------------------------------------
;       Unterprogramme und Interrupt-Behandlungsroutinen
;-----------------------------------------------------------------------------
;Interruptroutine zum Senden und Empfangen über den SPI Bus
ISRSPI: in      r16,PINA        ;Schalter einlesen und ueber SPI versenden
        out     SPDR,r16
        in      r16,SPDR        ;Empfange Byte vom Slave an den LEDs ausgeben
        out     PORTC,r16
        reti                    ;Rücksprung ins Hauptprogramm
```

Programm für den Slave:

```
;-----------------------------------------------------------------------------
;       Sprungadressen fuer die Interrupts organisieren (ISR VECTORS)
;-----------------------------------------------------------------------------
        ;Vektortabelle (im Flash-Speicher)
.ORG   SPIaddr                  ;interner Vektor fuer SPI (alt.: .ORG 0x0018)
       rjmp    ISRSPI           ;Springe zur ISR von SPI Transfer Ready
```

...

```
        ;Schalter und LEDs initialisieren
        clr     Tmp1            ;8 Schalter an PORTA
        out     DDRA,Tmp1
        ser     Tmp1
        out     PORTA,Tmp1      ;Pull-Ups
        out     DDRC,Tmp1       ;8 LEDs an PORTC

        ;SPI Slave initialisieren
        ;SPI-Ausgang initialisieren
        ;(PB4(/SS), PB5(MOSI) und PB7(SCK) automatisch Eingang)
        sbi     DDRB,6          ;PB6 Ausgang (MISO)
        ;SPI einschalten, Slave mit SCK = Systemtakt/128, MSB first, SPI Mode 1
```

```
        ldi     r16,0xC7        ;SPCR = 0b11000111
        out     SPCR,r16        ;B7 SPIE=0 kein Interrupt
                                ;B6 SPE=1 SPI einschalten
                                ;B5 DORD=0 MSB zuerst
                                ;B4 MSTR=0 Slave
                                ;B32 CPOL=0 CPHA=1 SPI Modus 1
                                ;B10 SPR1=0 SPR0=1 Systemtakt/128 (125kHz)

;-----------------------------------------------------------------------------
;       Hauptprogramm
;-----------------------------------------------------------------------------
MAIN:   sei                     ;Interrupts Global erlauben
END:    rjmp    END             ;Endlosschleife

;-----------------------------------------------------------------------------
;       Unterprogramme und Interrupt-Behandlungsroutinen
;-----------------------------------------------------------------------------
;Interruptroutine zum Senden und Empfangen über den SPI Bus
ISRSPI: in      r16,PINA        ;Schalter einlesen und
        out     SPDR,r16        ;ueber SPI uebertragen
        in      r16,SPDR        ;Empfangene Daten auf LEDs ausgeben
        out     PORTC,r16
        reti                    ;Rücksprung ins Hauptprogramm
```

✎ **C401** Teste die Übertragung mit den beiden vorgegebenen Programme[19]. Schreibe die Programme so um, dass die Initialisierung jeweils in einem Unterprogramm erfolgt.
Nenne die neuen Programme "**C401_SPI_master_interrupt.asm**" und "**C401_SPI_slave_interrupt.asm**".

Weitere Aufgaben

✎ **C402** Im SRAM eines Slave befinden sich 10 Werte eines Sensors. Ein Master aktiviert den Slave um die 10 Werte abzufragen (Polling). Dabei wird für jedes Byte die /SS-Leitung aktiviert. Der Master speichert die 10 Werte im SRAM und gibt sie dann auf dem Display aus. Der Slave übermittelt die 10 Werte mit Hilfe einer Interruptroutine.
Nenne die neuen Programme "**C402_SPI_master_polling_10.asm**" und "**C402_SPI_slave_interrupt_10.asm**".

✎ **C403** Die folgende Aufgabe stammt aus dem Lehrbuch von Roland Walter (www.rowalt.de). Das Brenngerät (Programmer) dient in dieser Aufgabe als Master. Damit unser Slave nicht umprogrammiert wird, sondern die Daten über SPI richtig empfängt muss die Reset Leitung des Programmiergerätes mit dem /SS-Pin statt dem Reset-Pin des AVR verbunden werden. Bastle dazu einen Adapter. Die Befehle des Programmiergerätes können dann eingelesen werden und über die serielle Schnittstelle an einen PC übermittelt werden. Weitere Informationen zu den Befehlen findet man im Datenblatt des ATmega32A (bzw. ATmega8A) unter *Memory Programmimg, Serial Downloading, SPI Serial Programmimg Instruction Set.* Teste das Programm und finde heraus welche Befehle das Programmiergerät sendet. Nenne das neue Programm "**C403_SPI_slave_interrupt_programmer.asm**".

19 Download auf www.weigu.lu/a/asm

✎ **C404** Die folgende Aufgabe stammt ebenfalls aus dem Lehrbuch von Roland Walter (www.rowalt.de). Es soll ein Master so programmiert werden, dass er einen beliebigen anderen Controller als Brenngerät (Programmer) anspricht. Der Slave wird dabei vom Master mit Spannung versorgt. Seine Reset-Leitung wird über einen Ausgang (z.B. PB4,/SS) aktiviert. Die Befehle für das Programmiergerätes findet man im Datenblatt des ATmega32A (bzw. ATmega8A) unter *Memory Programmimg, Serial Downloading, SPI Serial Programmimg Instruction Set*. Das Programm soll den Slave in den Programmiermodus versetzen, dann die drei Signatur-Bytes auslesen und dann den Programmiermodus wieder beenden. Nenne das Programm "**`C404_SPI_master_polling_programmer.asm`**".