Lösung 08: UART, SPI, I2C

Aufgabe 1: UART per AVR-Libc

a) Es gilt: $Baudrate = \frac{f_{osc}}{16(UBBRn+1)} \rightarrow UBBRn = \frac{f_{osc}}{Baudrate \cdot 16 \cdot} - 1 = \frac{16MHz}{9600\frac{1}{s} * 16} - 1 = 103$

Dieser Wert wird im UBBR**0** Register konfiguriert, da wir USART**0** verwenden. Der Zähler zählt jeweils von diesem Wert herunter. Bei Erreichen der 0 wird je ein Takt erzeugt, wobei dieser Takt im Asynchronous Normal Mode nochmals durch 2, 4 und 2 geteilt wird (insgesamt also 16), siehe Abbildung 22-2. Daraus erklärt sich die Formel in Tabelle 22-1.

.

b) Achtung: Der ATmega2560 hat 4 USART-Module. Alle Makros müssen deshalb z.B. UBBR0L, UDR0, etc. heißen, falls USART0 verwendet wird.

```
#define FOSC 16000000
#define MYUBBR 103

 void setup()
 {

        // set baud rate
        UBRR0L = (unsigned char) MYUBRR;
        UBRR0H = (unsigned char) (MYUBRR >> 8);

        // Enable transmitter p220
        UCSR0B = (1<<TXEN0);

        // Set frame format: 8 data, 1 stop bit, p221
        UCSR0C =  (1<<UCSZ01) | (1<<UCSZ00);

 }


 void uart_putchar(char c) {

        // Wait until data register empty, p207 and p219
        while (!(UCSR0A & (1 << UDRE0)));
        UDR0 = c;
 }

 void loop()
 {
        char text[] = "Hallo";
        int i = 0;
        for (i = 0; i < sizeof(text); i++)  {
               uart_putchar(text[i]);
        }
        delay(2000);
 }
```

Aufgabe 2: SPI

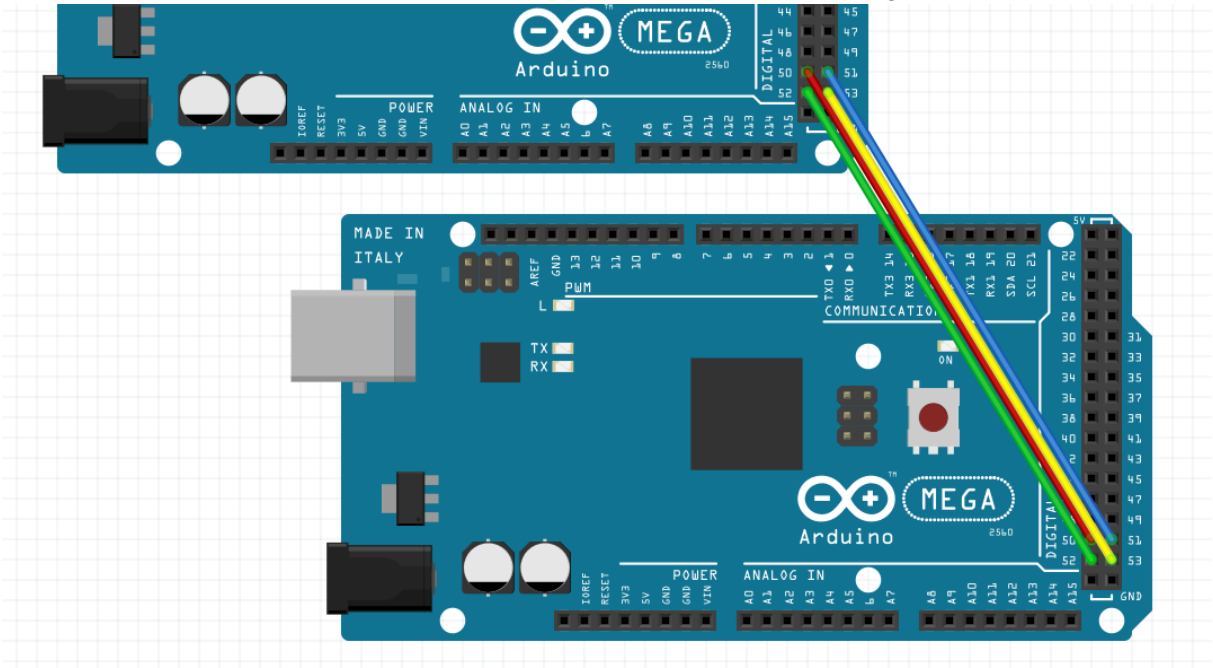
a) Es gilt die folgende Zuordnung. Die rechte Spalte zeigt die Buchsen und gibt an, wie die Kabel angesteckt werden müssen.

MOSI	PB2	Digital Pin 51

MISO	PB3	Digital Pin 50
SCK	PB1	Digital Pin 52
SS	PB0	Digital Pin 53

b) Man benötigt 4 Verbindungskabel, siehe Zeichnung. Wichtig ist vor allem, dass man die Kabel nicht „überkreuzt“. Man darf MOSI des Masters nicht in MISO des Slaves stecken. Bei einer UART-Schnittstelle dagegen wäre es so, dass der TxD Ausgang mit dem RxD Eingang des Kommunikationspartners verbunden wird.
Zusatzinfos:

- Die Schreibweise $\overline{SS}$ ist ein Hinweis auf negative Logik. Wenn LOW anliegt, dann wird ein Slave ausgewählt. Ohne weitere HW-Logik könnte ein Master mehrere Slaves ansteuern, in dem er immer nur einen gezielt durch LOW auswählt (negative Logik) und alle anderen durch HIGH deaktiviert.
- Gesendet wird aber erst, wenn der Master einen Takt an SCK anlegt.



c) SPI ist eine Vollduplex-Schnittstelle, d.h. zwischen Master und Slave werden **gleichzeitig** Daten in beide Richtungen übertragen. Die setup-Funktion unterscheidet sich deutlicher, die `spi_transceive`-Funktion ist für Master und Slave fast identisch.

Code für Master:

```
void setup() {
	Serial.begin(9600);

	// data direction: set MOSI==DDB2 and SCK==DDB1 and SS=DDB0 to output p96ff
	DDRB = (1<<DDB2) | (1<<DDB1) | (1<<DDB0);

	// Master init: Enable SPI, set as master, set clock rate to fck/128 p197/198
	SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPR1);
}

// send and receive data
unsigned char spi_transceive(unsigned char data) {
```

```
        // set SS to low, activating slave, synchronization (to be on safe side)
        PORTB &= ~(1<<DDB0);

        // start transmission by putting data into buffer
        SPDR = data;

        // wait until transmission completes
        while(!(SPSR & (1<<SPIF)));

        // return received data
        char result = SPDR;

        // set SS to high, deactivate slave
        PORTB |= (1<<DDB0);

        return result;
}

void loop() {
        char text[] = "Hallo Slave!";
        for (int i = 0; i < sizeof(text); i++)  {
                char received = spi_transceive(text[i]);
                Serial.print(received);
        }
        delay(1000);

}
```

Wichtig: Hier findet tatsächlich **Vollduplex-Datenübertragung** statt. Der Master legt die zu sendenden Daten in den Puffer SPDR. Nach dem Beenden der Übertragung liegen im Puffer wie von Geisterhand die vom Slave empfangenen Daten. Der Slave kann nur senden, wenn der Master etwas sendet. Aber Datenverkehr in beide Richtung ist gleichzeitig möglich.

Code für Slave:

```
void setup() {
        Serial.begin(9600);

        // data direction: set MIS==DDB3 to output (p96ff), all other input
        DDRB = (1<<DDB3);

        // Slave Init: Enable SPI, not setting MTSR means that it is a slave, p197/198
        SPCR = (1<<SPE);
}


// send and receive data
unsigned char spi_transceive(unsigned char data) {

        // start transmission by putting data into buffer
        SPDR = data;

        // wait until transmission completes
        while(!(SPSR & (1<<SPIF)));

        // return received data
        return (SPDR);
}

void loop() {
        char text[] = "Hallo Slave!";
```

```
        for (int i = 0; i < sizeof(text); i++)  {
                char received = spi_transceive(text[i]);
                Serial.print(received);
        }
        delay(1000);

}
```

Aufgabe 3: SPI vs. I2C

Vergleichen Sie SPI und I2C bzgl. der folgenden Kriterien:

- Overhead: Nutzdaten im Vergleich zu Overhead.
  Die Größe des SPI-Schieberegisters ist 8 Bit. Es gibt keinerlei Kontroll- oder Metadaten in einem Frame. Rein theoretisch besteht jede Übertragung nur aus Nutzdaten → 100% Nutzdaten und 0% Overhead. Die zur Verfügung stehende Bandbreite ist aber natürlich begrenzt durch die Taktrate mit der die Bits vom Sender zum Empfänger bzw. umgekehrt geschoben werden.
  Bei I2C gibt es Kontrolldaten. Annahme es werden 8 Bit übertragen, dann fallen z.B. folgende Kontrolldaten an, siehe Vorlesung: 1x Startbit, 7 x Adressbits, 1x R/W, 2x ACK, P/SR = 12 Bits Overhead!

- Anzahl der benötigten Verbindungsleitungen/kabel: Bei I2C 2x (SDA, SDL), bei SPI 3x (MOSI, MISO, SCK, $\overline{SS}$). SS wird nicht zwingend benötigt. Man könnte den Slave dauerhaft auswählen, indem man $\overline{SS}$ einfach immer auf LOW hält.

- Vollduplex, halbduplex, simplex: Mit SPI kann man vollduplex übertragen. I2C ist halbduplex.

- Quittierung für empfangene Daten: Nur bei I2C

Wann setzt man was ein? Das kann man nicht so einfach sagen. Häufig werden Speichersteine eher über SPI angebunden, da etwas schneller. Zuverlässiger ist die Datenübertragung z.B. wegen ACKs bei I2C. Letzteres findet man öfters zur Abfrage von Sensoren. Bei I2C kann man auch mit einem Bus mehrere Sensoren gleichzeitig abfragen.