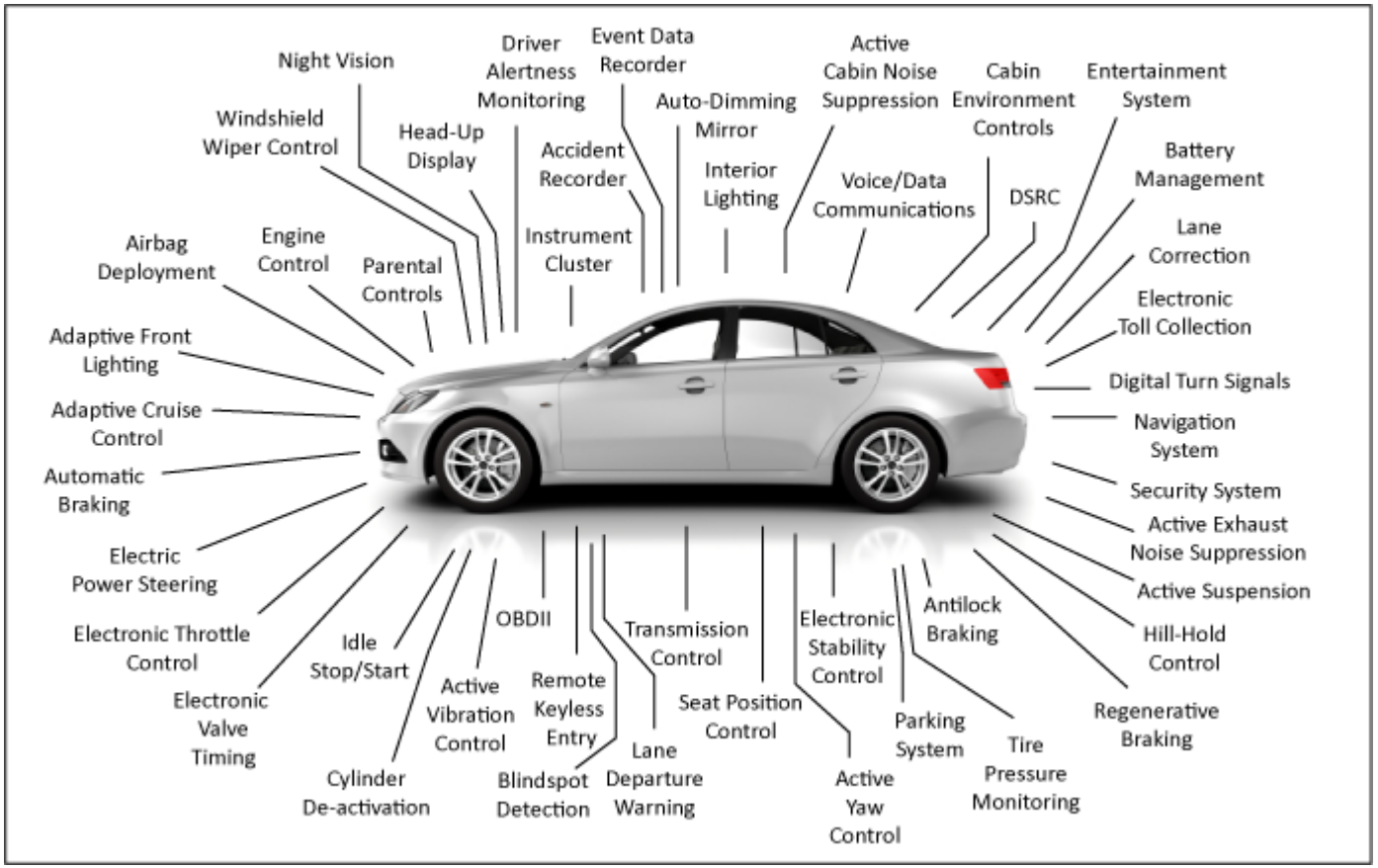
Technische Hochschule Rosenheim
Technical University of Applied Sciences

Embedded Systems
Kapitel 8: Kommunikationsschnittstellen

Prof. Dr. Wolfgang Mühlbauer
Fakultät für Informatik
wolfgang.muehlbauer@th-rosenheim.de

Sommersemester 2020

Motivation



Quelle: [4]

- Wie kommuniziert Mikrocontroller mit
  - anderen Mikrocontrollern?
  - Peripherie, Sensoren und Aktoren?

→ Standardisierte Kommunikationsschnittstellen

Kommunikation: Klassifizierung (1)

- **Seriell vs. parallel**
  - Parallel == *Gleichzeitiges* Übertragen mehrerer Bits
  - Benötigt mehrere Datenleitungen
  - Höherer Durchsatz?

- **Synchron vs. asynchron**
  - *Synchron*: Sender und Empfänger haben gemeinsame Uhr
    - Meist eigene Datenleitung für Takt.
  - *Asynchron*: Keine gemeinsame Uhr
    - Konfiguration der Takt- bzw. Baudrate auf beiden Seiten
    - Oversampling: Empfänger tastet mit höherer Frequenz ab
    - Erkennen des Übertragungsbeginns durch spezielle Symbole (Start-/Stopbit).

- **Bus vs. Point-to-Point**
  - *Bus*: Mehr als 2 Geräte können sich gegenseitig hören.
  - Erfordert Adressierung.

Kommunikation: Klassifizierung (2)

- **Vollduplex vs. halbduplex**
  - *Vollduplex*:
    - Datenübertragung in beide Richtungen *gleichzeitig*
    - Erfordert separate Leitungen für Senden und Empfangen.
  - *Halbduplex*
    - Vielfachzugriff (Multiple Access Control), siehe Rechnernetze!

- **Peer-to-Peer vs. Master-Slave**
  - *Master:* Nur 1 Seite (== Master) darf die Kommunikation starten.

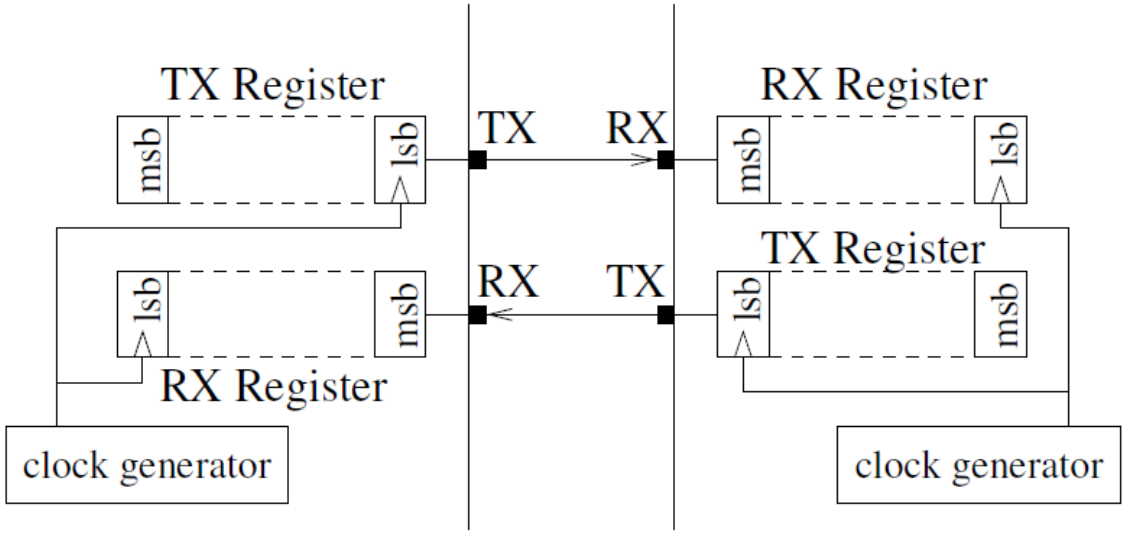
- **Differential vs. Single-Ended**
  - *Single-ended*
    - 1 gemeinsame GND Leitung
    - Alle Spannungspegel sind bezogen auf gemeinsames GND
    - Problematisch bei großen Entfernungen → Rauschen!
  - *Differentiell*
    - Spannungsunterschied zwischen 2 Leitungen trägt Information.
    - Übertragung erfordert 1 Leitungspaar für jede Datenübertragung.

Inhalt

- **UART**
- SPI
- I^2C
- 1-Wire

Universal **Asynchronous** Receiver Transmitter (UART)

- Alternativer Name: *Serial Communication Interface* (SCI)
- Sehr weit verbreitet
- Eigenschaften
  - Asynchron
  - Seriell
  - Vollduplex (meistens): TxD und RxD
- Übertragungsparameter ***müssen konfiguriert*** werden.



Datenleitungen:
- TX/TXD: Senden
- RX/RXD: Empfangen

→ *"2 wires" (asynchron)*

Quelle: [1]

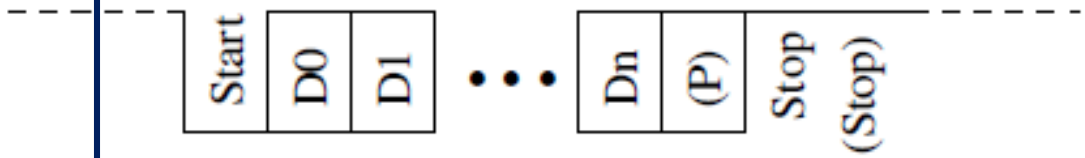
Konfigurationsparameter

- ***Anzahl Datenbits (D)*** pro Frame
  - Zwischen 5 und 9 Bits
- ***Paritätsbit***
  - Soll Parität verwendet werden? (*N*: No Parity)
  - Gerade oder ungerade Parität (*E* oder *O*)?
- ***Stop Bit (S)***
  - 1 oder 2 Stop-Bits am Ende der Übertragung?
- ***Baudrate*** („Symbolrate)
  - Sender und Empfänger müssen beide gleiche Baudrate konfigurieren.
  - Hohe Baud-Rate: Hoher Durchsatz!

Nomenklatur:
D{E|O|N}S

Beispiel: 8E1
→ 8 Datenbits, gerade Parität, 1 Stopbit

UART Frame

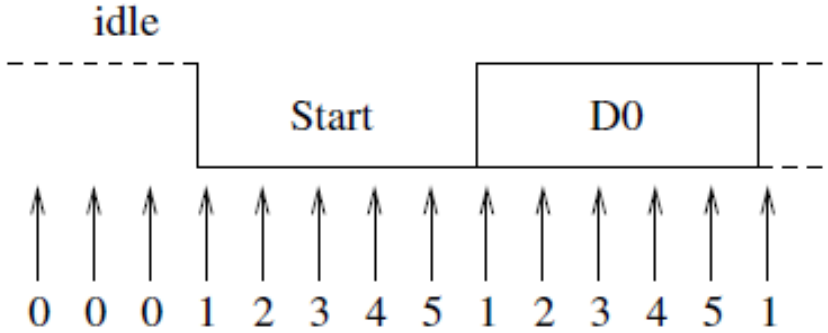


Quelle: [1]

Synchronisation, Erzeugung der Baud-Rate

Synchronisation

- Start Bit: Empfänger muss fallende Flanke erkennen
- Abtastrate des Empfängers deutlich höher als Datenrate (*Oversampling*)



UART Bit Sampling

Quelle: [1]

Erzeugung der Baud-Rate

- Hinweis: Baudrate muss dem Empfänger bekannt sein!
- Von Systemtakt abgeleitet + Timer + Prescaler
  - → Nicht jede Baud-Rate wird unterstützt

UART beim ATmega2560

- **USART-Module**
  - Asynchrone + synchrone Kommunikation möglich.
    - **Asynchron**: Klassisches UART mit unterschiedlichen Geschwindigkeiten „*normal asynchronous*“ und „*double speed asynchronous*“.
    - **Synchron:** Master oder Slave kann Takt vorgeben. Ähnelt dann SPI!
  - 4mal vorhanden
    - USART0, USART1, USART2, USART3

- **Baud Generation**
  - Programmierer definiert maximalen Zählerwert in Register `UBBR`.
  - Zähler startet mit diesem Wert, dekrementiert mit Systemtakt.
  - Bei Zählerstand 0: Sende Symbol („Baudrate“) und setze zurück auf `UBBR`.
  - Langsamere Baudrate durch weitere Prescaler möglich.

- **Arduino Mega Board:** Zugriff per USB nur auf `U(S)ART0`
  - Entwicklerboard besitzt weiteren Controller Atmega16U2
  - Dieser setzt UART in USB um.
  - PC sieht das als COM-Port.

USART Register

- **UDR**
  - In C: Bei Lesezugriff empfangenes Byte, bei Sendezugriff zu sendendes Byte.
  - Hardware-intern: 2 Register, ein Lese- und ein Senderegister (Handbuch, S.218)

- **UCSRnA**
  - Infos zur Übertragung.
  - Beispiel: Wurde Übertragung erfolgreich beendet?

- **UCSRnB**
  - USART-bezogene Interrupts.
  - Aktivieren des Empfängers und Receivers.

- **UCSRnC**
  - Auswahl des Modus (asynchron oder synchron)
  - Datenformat: Stoppbit, Parität?

- **UBRRnL**
  - Einstellen der Baudrate

UART mit der Arduino Bibliothek

- Arduino Library
  - https://www.arduino.cc/en/Reference/Serial
- ATmega2560 verfügt über 4 U(S)ART Schnittstellen
  - Zugreifbar über `Serial, Serial1, Serial2, Serial3`

Beispiel:
Programmierung des U(S)ART mit Arduino Bibliothek

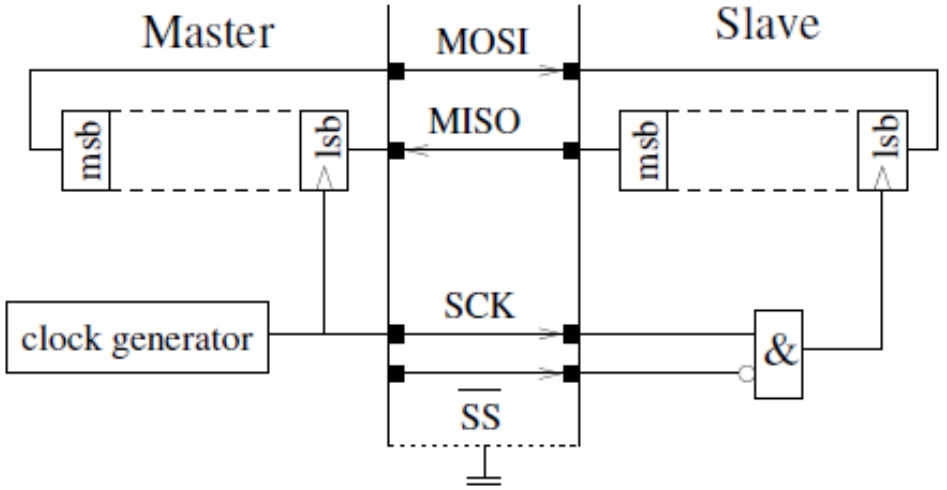
```
byte byteRead;

void setup() {
  // Turn the Serial Protocol ON
  Serial.begin(9600);
}

void loop() {
  /*  data available for read */
  if (Serial.available()) {
    /* read the most recent byte */
    byteRead = Serial.read();
    /*ECHO read value back to the serial port. */
   Serial.write(byteRead);
}}
```

Inhalt

- UART
- **SPI**
- I^2C
- 1-Wire

Serial Peripheral Interface (SPI)

- **Eigenschaften**
  - **Synchron**, Vollduplex, Single-ended
- **Master-Slave**
  - Master bestimmt Takt.
  - In jeder Taktperiode Übertragung eines Bytes.
  - Slave muss vor Übertragung durch $\overline{SS}$ = 0 aktiviert werden.
- **Datenübertragung:** In jedem Taktzyklus
  - `MOSI:` 8 Bits von Master-Schieberegister in Slave-Schieberegister
  - `MISO:` 8 Bits von Slave-Schieberegister in Master-Schieberegister



Datenleitungen:
- `MOSI`: Master Out, Slave In
- `MISO`: Master In, Slave Out
- `SCK`: System Clock
- [$\overline{SS}$: Slave Select]

→ *3 wires (synchron)*

Quelle: [1]

SPI beim ATmega2560

- **Nur 1 SPI Einheit**
  - Kapitel 21, S. 190
  - Aber: Jede USART Einheit kann als SPI eingesetzt werden.
- Datenübertragung sobald Master Takt auf SCK legt
  - Slave kann nur Daten senden, wenn Master etwas sendet.
- Nicht vergessen: Slave durch dauerhaftes oder vorübergehendes $\overline{SS}$=0 aktivieren
- **Register**
  - **SPCR:** SPI Control Register
    - Konfiguration: Aktivierung, Interrupts, Master oder Slave?
    - Daten bei steigenden oder fallenden Flanken lesen?
  - **SPSR:** SPI Status Register
    - Informationen
    - Beispiel: trat SPI Interrupt auf?
  - **SPDR:** SPI Data Register
    - Enthält zu sendende und empfangende Daten
    - Vollduplex: Nach Ablauf einer Taktperiode sind 8 Bits aus dem Register gesendet worden und die 8 empfangenen Bits stehen nun in diesem Register.

SPI mit der Arduino-Bibliothek

- **Arduino-Bibliothek** für SPI-Schnittstelle:
  - https://www.arduino.cc/en/Reference/SPI
- **Beispielcode:**
  - Master überträgt Zeichenkette "*Fab*" vom Arduino zum Slave
  - Hier: Nur unidirektional Daten vom Master zum Slave.
- **Achtung!!**
  - Arduino Library `SPI` unterstützt keine Konfiguration als Slave.

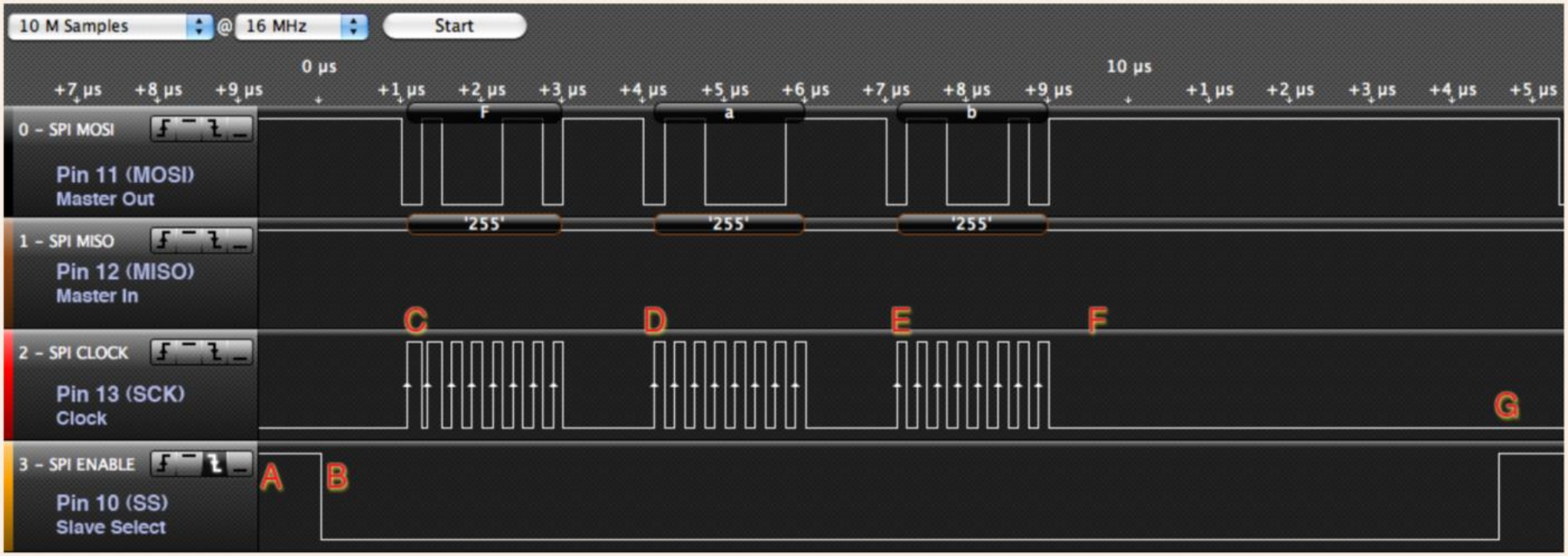
```
#include <SPI.h>

void loop (void)
{
  digitalWrite(SS, HIGH);  // ensure SS stays high

// Put SCK, MOSI, SS pins into output mode
// also put SCK, MOSI into LOW state, and SS into HIGH state.
// Then put SPI hardware into Master mode and turn SPI on
SPI.begin ();

delay (5000);  // 5 seconds delay to start logic analyser.

char c;

// enable Slave Select
digitalWrite(SS, LOW);    // SS is pin 10

// send test string
for (const char * p = "Fab" ; c = *p; p++)
   SPI.transfer (c);

// disable Slave Select
digitalWrite(SS, HIGH);

// turn SPI hardware off
SPI.end ();

while (1);  //loop
}
```

Quelle: [5]

Aufzeichnung einer SPI Kommunikation

- Logikanalysator zeichnet Signalverlauf auf.
  - **B**: Slave wird durch $\overline{SS}$=0 aktiviert. Man könnte ihn auch dauerhaft aktiv lassen.
  - **C**: Master beginnt Clock zu erzeugen. Es werden 8 Bit übertragen → Zeichen "F"
  - **D** und **E**: Zeichen "A" und "B"
  - **F**: Master erzeugt keinen Takt mehr. Keine Datenübertragung.

Quelle: [1]



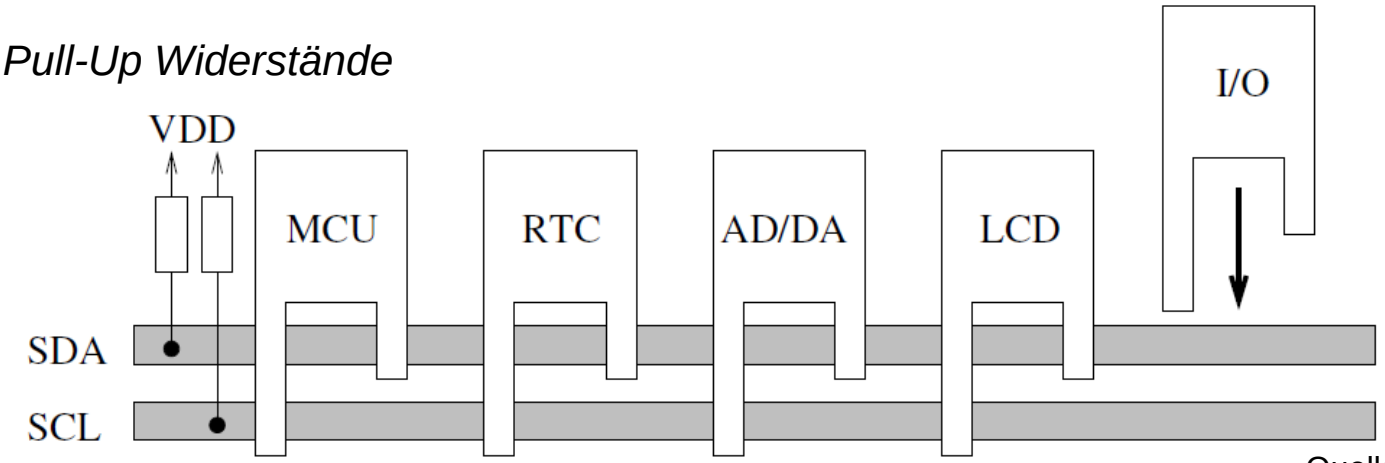
Inhalt

- UART
- SPI
- **I2C**
- 1-Wire

Inter-IC Bus (I^2C)

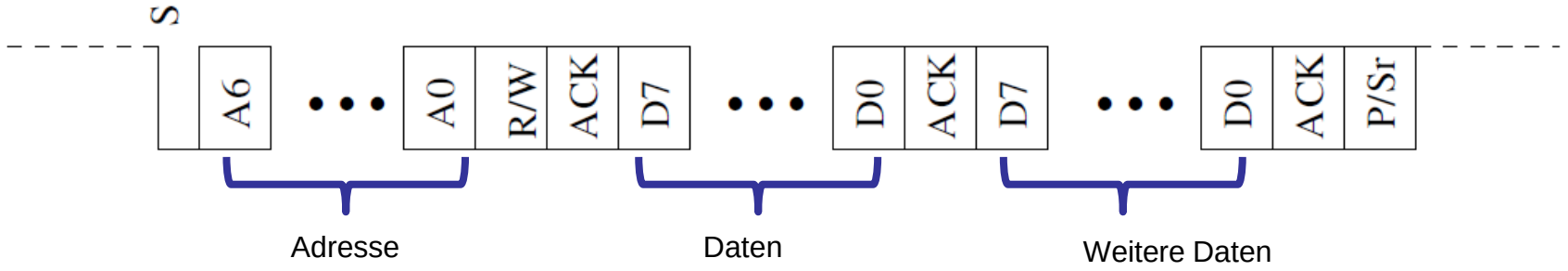
- ***Eigenschaften***
  - Synchron, Bus, Single-Ended, Halbduplex, Master Slave
- ***Adressierung***
  - Meist 7-Bit Adresse, Unterstützung von bis 120 externer Geräte
- ***Datenleitungen***
  - SCL: Serial Clock Line
  - SDA: Serial Data Line
  - -> "2 wires" (synchron). Deshalb oft auch **Two-Wire Interface (TWI)** genannt.

Pull-Up Widerstände



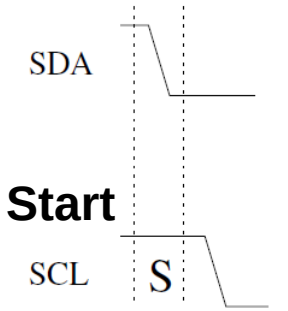
Quelle: [1]

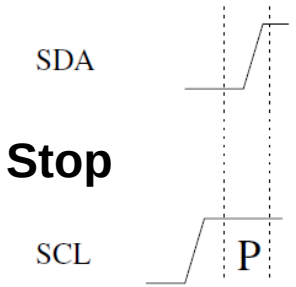
Ablauf der Datenübertragung



Quelle: [1]

- ***Startbedingung:*** Fallende Flanke von `SDA` während `SCL==HIGH`
- Anlegen der ***Adresse*** der Gegenstelle
- R/$\overline{W}$: Master spezifiziert ob ***Lese- oder Schreibzugriff***
  - Entsprechender Slave bestätigt Kenntnisnahme durch ACK.
- ***Datentransfer***:
  - Übertragung mehrere Bytes möglich.
  - Empfänger quittiert jedes einzelne Byte.
- ***Stoppbedingung:*** Steigende Flanke von `SDA` während `SCL==HIGH`





Quelle: [1]

ATmega2560, I²C mit der Arduino Bibliothek

- **ATmega2560**
  - TWI-Interface, siehe Kapitel 24.
  - Kann Master oder Slave sein.
  - Maximale Übertragungsrate: 400 kHz.

- **Programmierung:**
  - ***AVR-Libc***
    - Recht komplex: Programmierer muss Signalfolgen selbst erzeugen.
    - Datenblatt, Seite 246.
  - ***Arduino Library***
    - `Wire` Library: https://www.arduino.cc/en/Reference/Wire
    - Relativ einfache Handhabung, komfortabel.
    - Siehe Live Coding.

Live Coding: I²C mit Wire Library

- ***Anforderung***
  - Arduino Uno: I²C Slave und Transmitter, der „*Greetings from slave*“ sendet.
  - Arduino Mega: I²C Master und Receiver
- ***Vorgehen***
  - Verbinde `SDA`, `SCL` und `GND` der beiden Mikrocontroller
  - Master: `Wire.begin()`, d.h. keine Angabe einer Adresse
  - Slave: `Wire.begin(8)`, d.h. Slave hat Adresse 8

```
#include <Wire.h>

void setup() {
   Wire.begin();         // join i2c bus as master
   Serial.begin(9600);   // start serial for output
}

void loop() {
   Wire.requestFrom(8, 21);     // request 6 bytes from slave device #8
   while (Wire.available()) {   // slave may send less than requested
      char c = Wire.read();     // receive a byte as character
      Serial.print(c);          // print the character
   }
   delay(500);
}
```

Code für Master.

Wie sieht der dazu-Gehörige Code für den Slave aus?

Zusammenfassung

	UART	**SPI**	**I²C**
Seriell	Ja	Ja	Ja
Duplex	Ja	Ja	Nein
Synchron	Nein	Ja	Ja
Bus	Nein	Jein	Ja
Anzahl Leitungen	2	3	2
Datenrate bei ATmega2560	$BAUD = \frac{f_{OSC}}{16(UBRRn+1)}$	f_{osc}/128 - f_{osc}/2	Max. 400 kbit/s

Hinweis: I3C is the "successor" of I2C
https://en.wikipedia.org/wiki/I3C_(bus)

Ausblick: 1-Wire

- Asynchrone, Halbduplex
- Nur 1 Datenleitung
- 1 Master, mehrere Slave
  - Slave hat fest einprogrammierte 64-Bit ID
- Datenübertragung
  - Normalerweise ist Bus immer auf HIGH → Pull-Up!
  - ***Logisch 1***: Ziehe Bus 1 bis 15 µs auf LOW.
  - ***Logisch 0:*** Ziehe Bus 60 bis 120 µs auf LOW.
- Datenleitung auch Stromversorgung für Slaves.
  - Kondensatoren in Slaves überbrücken kurze LOW Zeiten.
- Keine direkte HW-Unterstützung durch Atmega!
  - Aber möglich: http://www.atmel.com/images/Atmel-2579-Dallas-1Wire-Master-on-tinyAVR-and-megaAVR_ApplicationNote_AVR318.pdf

Quellenverzeichnis

[1] G. Gridling und B. Weiss. *Introduction to Microcontrollers*, Version 1.4, 26. Februar 2007, Kapitel 2.5, verfügbar online: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf (abgerufen am 08.03.2017)

[2] Datenblatt ATmega2560, http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf, (abgerufen am 19.03.2017)

[3] AVR-GCC Tutorial, https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial#Programmieren_mit_Interrupts (abgerufen am 02.04.2017)

[4] http://www.chipsetc.com/uploads/1/2/4/4/1244189/6873681_orig.png?319 (abgerufen am 19.05.2017)

[5] http://www.gammon.com.au (abgerufen am 24.05.2019)