

# Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2019/2020

## Übung 3

Benedict Herzog  
Bernhard Heinloth  
Tim Rheinfels

Lehrstuhl für Informatik 4  
Friedrich-Alexander-Universität Erlangen-Nürnberg



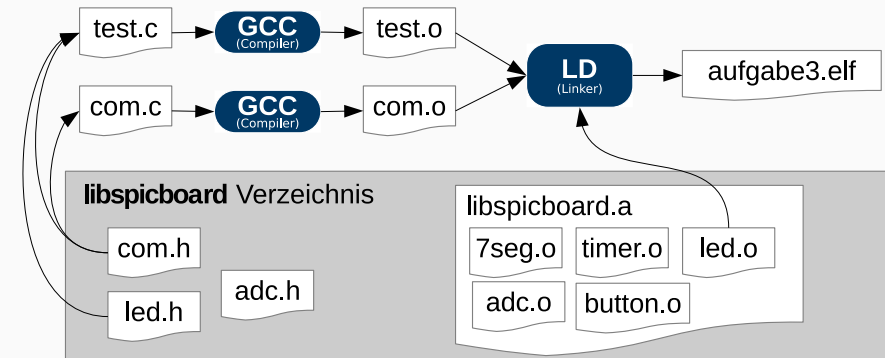
Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



## Vorstellung Aufgabe 1

## Module

## Ablauf vom Quellcode zum laufenden Programm



1. Präprozessor
2. Compiler
3. Linker
4. Programmier/Flasher



- Header Dateien enthalten die Schnittstelle eines Moduls
  - Funktionsdeklarationen
  - Präprozessormakros
  - Typdefinitionen
- Header Dateien können u.U. mehrmals eingebunden werden
  - led.h bindet avr/io.h ein
  - button.h bindet avr/io.h ein
  - ~ Funktionen aus avr/io.h mehrmals deklariert
- Mehrfachinkludierung/Zyklen vermeiden ~ **Include-Guards**
  - Definition und Abfrage eines Präprozessormakros
  - Konvention: Makro hat den Namen der .h-Datei, ' ' ersetzt durch '\_'
  - z.B. für button.h ~ BUTTON\_H
  - Inhalt nur einbinden, wenn das Makro noch nicht definiert ist
- **Vorsicht:** flacher Namensraum ~ möglichst eindeutige Namen

2

- Erstellen einer .h-Datei (Konvention: gleicher Name wie .c-Datei)

```

01 #ifndef LED_H
02 #define LED_H
03 /* fixed-width Datentypen einbinden (im Header verwendet) */
04 #include <stdint.h>
05
06 /* Datentypen */
07 typedef enum {
08     RED0    = 0,
09     YELLOW0 = 1,
10     [...]
11     GREEN1  = 6,
12     BLUE1   = 7
13 } LED;
14
15 /* Funktionen */
16 void sb_led_on(LED led);
17 [...]
18 #endif //LED_H

```

3



- Module müssen Initialisierung durchführen
  - zum Beispiel Portkonfiguration
  - **Java:** mit Klassenkonstruktoren möglich
  - **C:** kennt kein solches Konzept
- **Workaround:** Modul muss bei erstem Aufruf einer seiner Funktionen ggf. die Initialisierung durchführen
  - muss sich merken, ob die Initialisierung schon erfolgt ist
  - Mehrfachinitialisierung vermeiden
- Anlegen einer Init-Variable
  - Aufruf der Init-Funktion bei jedem Funktionsaufruf
  - Init-Variable anfangs 0
  - Nach der Initialisierung auf 1 setzen

- `initDone` ist initial 0
- wird nach der Initialisierung auf 1 gesetzt
- ~ Initialisierung wird nur ein mal durchgeführt

```

01 static void init(void){
02     static uint8_t initDone = 0;
03     if (initDone == 0) {
04         initDone = 1;
05         ...
06     }
07 }
08
09 void mod_func(void) {
10     init();
11     ...

```

4

5

Sichtbarkeit und Lebensdauer	nicht static	static
lokale Variable	Sichtbarkeit <b>Block</b> Lebensdauer <b>Block</b>	Sichtbarkeit <b>Block</b> Lebensdauer <b>Programm</b>
globale Variable	Sichtbarkeit <b>Programm</b> Lebensdauer <b>Programm</b>	Sichtbarkeit <b>Modul</b> Lebensdauer <b>Programm</b>
Funktion	Sichtbarkeit <b>Programm</b>	Sichtbarkeit <b>Modul</b>

- Lokale Variablen, die **nicht** static deklariert werden:  
 ~> auto Variable (automatisch allokiert & freigegeben)
- Funktionen als static, wenn kein Export notwendig

```

01 static uint8_t state; // global static
02 uint8_t event_counter; // global
03
04 void main(void) {
05     /* ... */
06 }
07
08 static void f(uint8_t a) {
09     static uint8_t call_counter = 0; // local static
10     uint8_t num_leds; // local (auto)
11     /* ... */
12 }

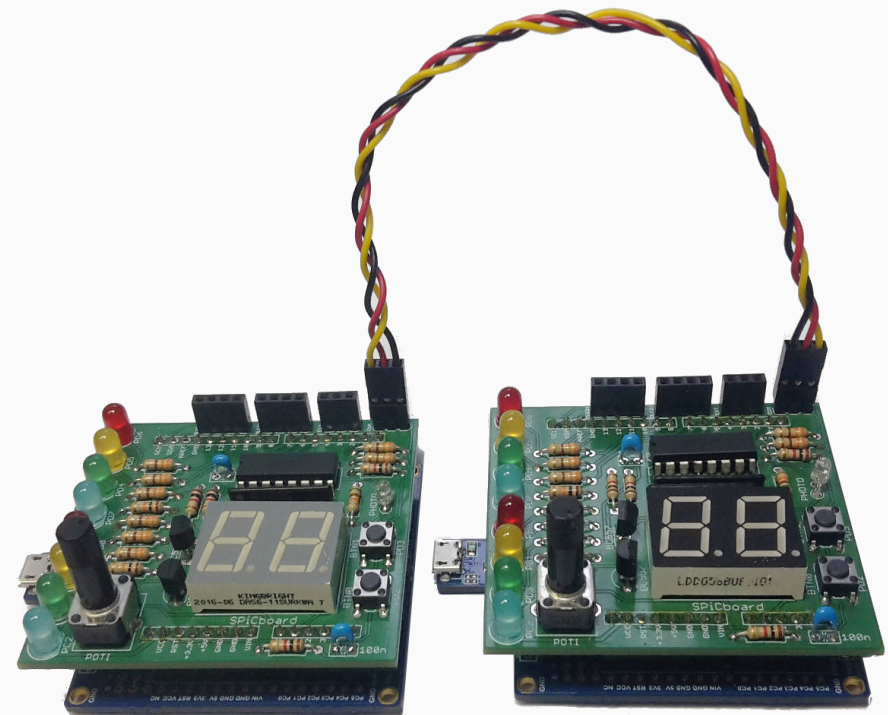
```

- Sichtbarkeit/Gültigkeit möglichst weit **einschränken**
- Globale Variable ≠ lokale Variable in f()
- Globale static Variablen: Sichtbarkeit auf Modul beschränken  
 ~> **wo möglich, static für Funktionen und globale Variablen**

6

7

## Kommunikation





**PD1** Ausgang (TX) wird mit RX des Kommunikationspartners verbunden

**PD0** Eingang (RX) analog mit Ausgang (TX) verbinden

**GND** wird mit GND verbunden

- gleiches Protokoll
  - wir nehmen 8-E-1
    - 8** Anzahl der Datenbits
    - E** gerades (*even*) Paritätsbit
    - 1** Stopbit
  - sowie (implizit) ein Startbit
  - das Datenbit mit dem niedrigsten Stellenwert wird zuerst übertragen (aufsteigend)
  - hoher Pegel entspricht einer logischen 1, niedriger Pegel einer 0
- gleiche Geschwindigkeit, bei uns 1200 Bd  
(1 Baud entspricht ein Symbol pro Sekunde)

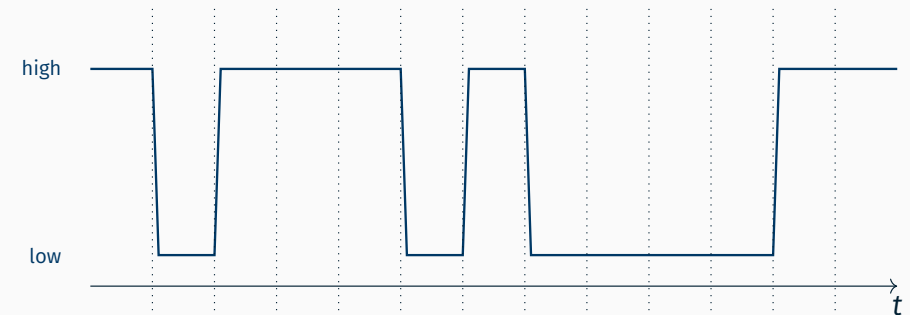
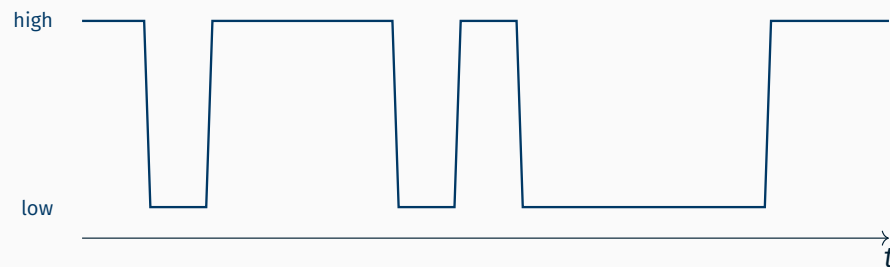
9

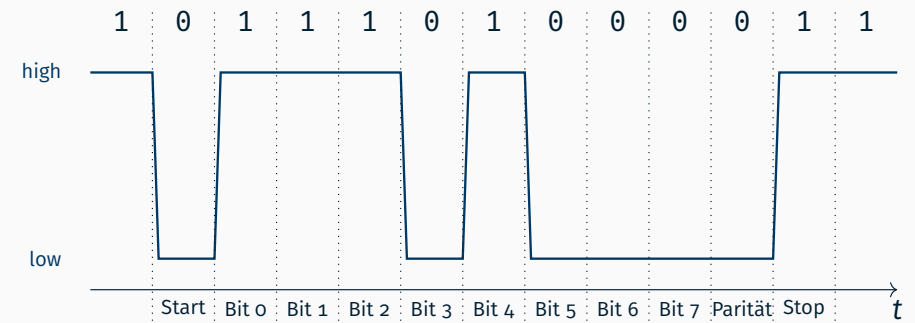
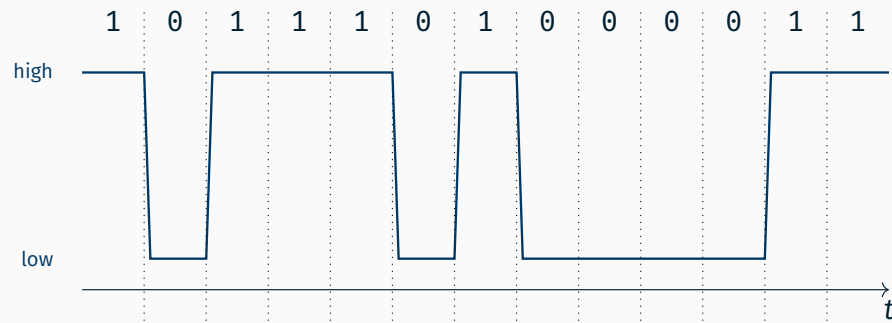
10

## Beispiel



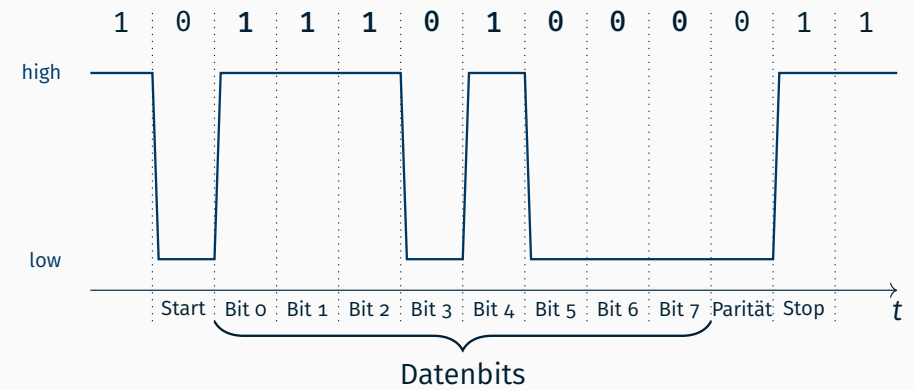
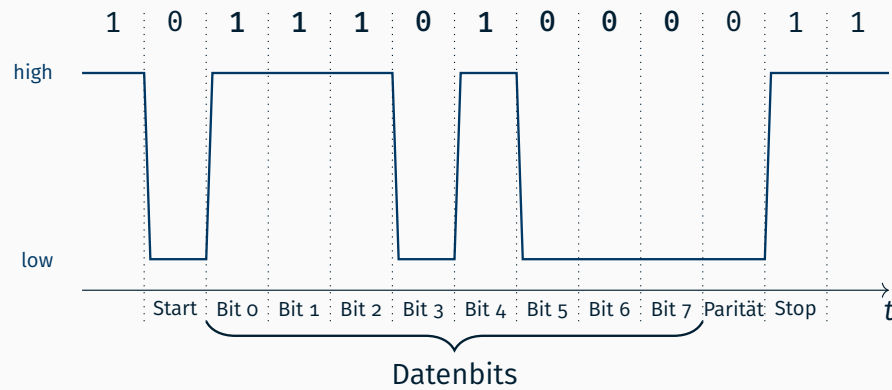
## Beispiel





11

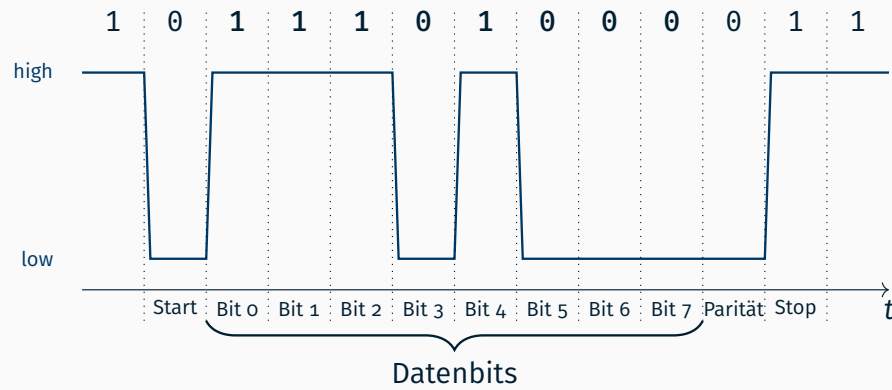
11



- Empfangenes Byte (rückwärts gelesen):  $00010111_2 = 0x17 = 23$

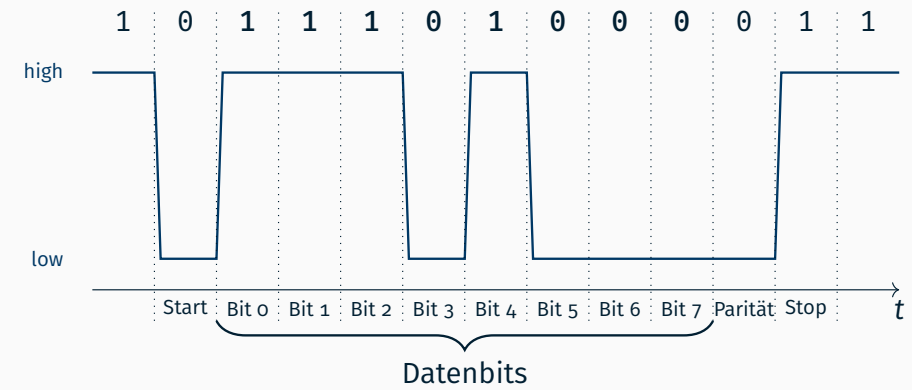
11

11



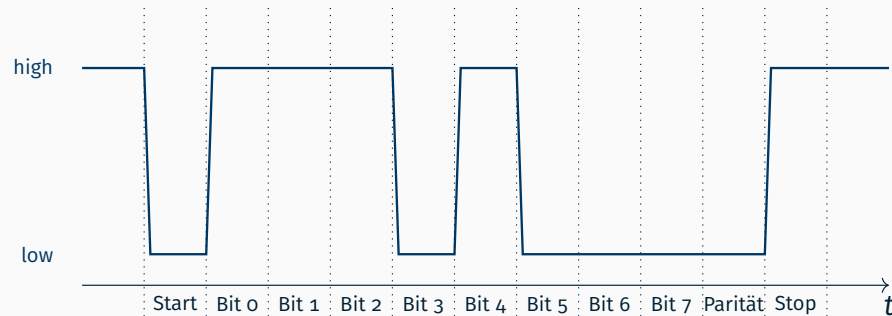
- Empfangenes Byte (rückwärts gelesen):  $00010111_2 = 0x17 = 23$
- Parität 0 (da Datenbits vier 1er  $\Rightarrow$  gerade Anzahl)

11

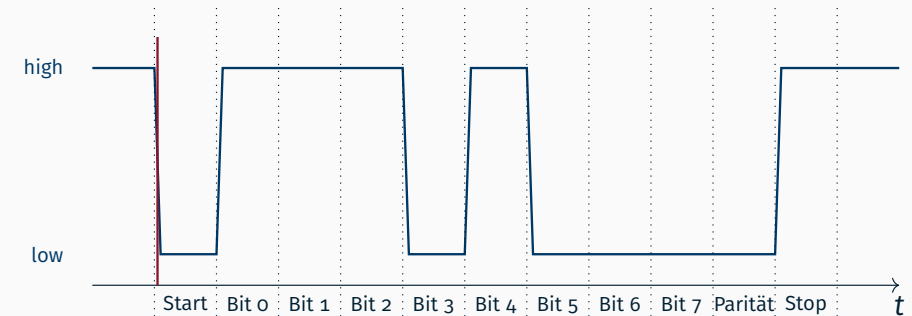


- Empfangenes Byte (rückwärts gelesen):  $00010111_2 = 0x17 = 23$
- Parität 0 (da Datenbits vier 1er  $\Rightarrow$  gerade Anzahl)
- Startbit immer 0 und Stopbit immer 1

11



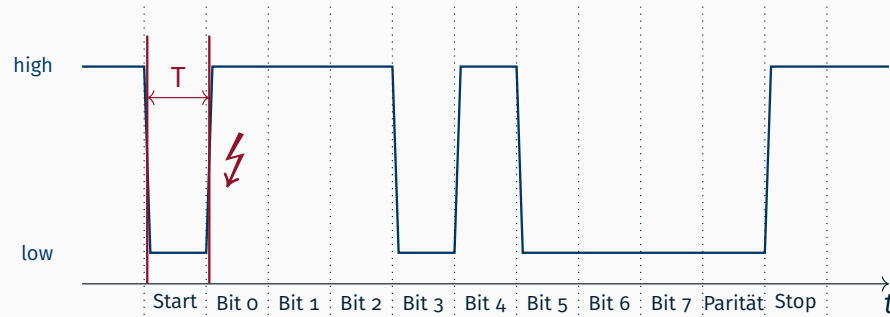
- Die Fenstergröße  $T$  ist uns bekannt



- Die Fenstergröße  $T$  ist uns bekannt
- Anfang ist einen Pegelwechsel von hoch auf niedrig

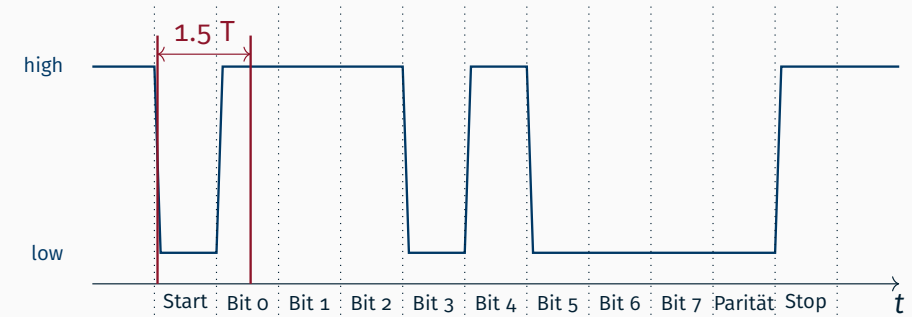
12

12



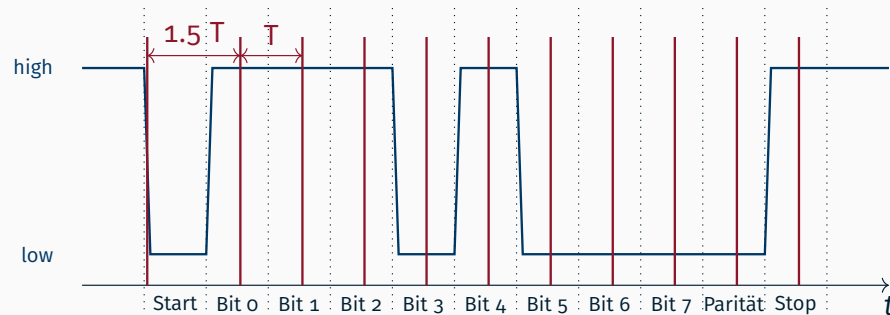
- Die Fenstergröße  $T$  ist uns bekannt
- Anfang ist einen Pegelwechsel von hoch auf niedrig

12



- Die Fenstergröße  $T$  ist uns bekannt
- Anfang ist einen Pegelwechsel von hoch auf niedrig
- warte danach  $1.5 T$  für ein eindeutiges Symbol

12



- Die Fenstergröße  $T$  ist uns bekannt
- Anfang ist einen Pegelwechsel von hoch auf niedrig
- warte danach  $1.5 T$  für ein eindeutiges Symbol
- für jedes weitere Symbol wieder alle  $T$  abtasten

12

## Aufgabe: Eigenes Kommunikationsmodule



- COM-Modul der SPiCboard-Bibliothek selbst implementieren
  - Modulschnittstelle bietet Funktionen für:
    - Versenden eines Bytes
    - Empfangen eines Bytes
  - Schnittstellendatei (com.h liegt im pub Ordner)
- Gleiches Verhalten wie das Original
- Beschreibung in der Doku auf der Webseite:

[https://www4.cs.fau.de/Lehre/WS19/V\\_SPIC/SPiCboard/libapi.shtml](https://www4.cs.fau.de/Lehre/WS19/V_SPIC/SPiCboard/libapi.shtml)

```
01 void sb_com_sendByte(uint8_t data);
02 uint8_t sb_com_receiveByte(uint8_t *data);
```

13

- Hilfsfunktionen zum Setzen/Auslesen der Register
  - Setzen des Pegels für das Senderegister TX
  - Auslesen des Pegels für das Empfangsregister RX
- Hilfsfunktionen gehören nicht zur Schnittstelle
  - ~> Sichtbarkeit einschränken
- Analog: Hilfsfunktion für die Initialisierung der Register
  - ~> sb\_com\_init()

```
01 static void sb_com_setTx(uint8_t bit);
02 static uint8_t sb_com_getRx();
```

14



- Funktionen der Schnittstelle müssen überall sichtbar sein
  - ~> Sichtbarkeit nicht einschränken
- Kompatibilität von Schnittstelle und Implementierung durch Inkludieren des Headers gewährleistet
  - Fehlercodes (COM\_ERROR\_STATUS)/Makros ebenfalls enthalten
  - Warum sind die Fehlercodes/Makros nicht in der .c Datei?

```
01 #include <adc.h>
02 #include <com.h>
03
04 void main(void) {
05     while(1) {
06         sb_com_sendByte(sb_adc_read(POTI) / 4);
07     }
08 }
```

15

- Aufgabe ist in drei Teile unterteilt
- Teilaufgabe a: Initialisierung
  - Einmaliges Initialisieren der Empfangs- und Senderegister
  - Implementieren der Hilfsfunktion sb\_com\_init()
  - Wird bei jedem Sende- oder Empfangsvorgang aufgerufen
  - Implementieren der Hilfsfunktionen sb\_com\_setTx() und sb\_com\_getRx()
  - Können benutzt werden, um den RX bzw. TX Pin zu lesen/setzen

16





- Aufgabe ist in drei Teile unterteilt
- Teilaufgabe b: Sendefunktionalität
  - Unabhängig vom Empfangen implementieren
  - Implementieren der Funktion `sb_com_sendByte()`
  - Ablauf für das zu sendende Byte:
    - Senden des Startbits
    - Senden der Nutzdaten (LSB zuerst)
    - Senden des Paritätsbits
    - Senden des Stoppbits
  - Verwendung von `sb_com_wait(1.0)` um ein Symbol zu warten
  - Testen der Sendefunktionalität mit dem vorgebenen Programm `test_receiver.elf`

16

- Aufgabe ist in drei Teile unterteilt
- Teilaufgabe c: Empfangsfunktionalität
  - Unabhängig vom Senden implementieren
  - Implementieren der Funktion `sb_com_receiveByte()`
  - Ablauf für das zu empfangende Byte:
    - Warten auf fallende Flanke an RX
    - Empfangen des Startbits
    - Empfangen der Nutzdaten
    - Empfangen des Paritätsbits
    - Empfangen des Stoppbits
  - Treten Fehler während der Übertragung auf, wird der Empfangsvorgang bis zum Ende ausgeführt und der entsprechende Fehlercode zurückgegeben
  - Testen der Empfangsfunktionalität mit dem vorgebenen Programm `test_sender.elf`

16



- Testen der finalen Implementierung mit einer Anwendung
- Beispielanwendung auf100
  - 2-Personen-Spiel zum Testen des Kommunikationsmoduls  
Spielprinzip: Es wird mit einer zufälligen Zahl begonnen, die auf der 7-Segmentanzeige dargestellt wird. Jeder Spieler kann diese Zahl nun abwechselnd durch drehen des Potentiometers um 1 bis 8 erhöhen. Der Spieler, der die 100 erreicht hat gewonnen.

## Hands-on: Module, Felder & Zeiger

---

17



- Statistikmodul und Testprogramm
- Funktionalität des Moduls (Schnittstelle):

```
01 // Schnittstelle
02 uint8_t avgArray(uint16_t *a, size_t s, uint16_t *avg);
03 uint8_t minArray(uint16_t *a, size_t s, uint16_t *min);
04 uint8_t maxArray(uint16_t *a, size_t s, uint16_t *max);
05
06 // interne Hilfsfunktionen
07 uint16_t getMin(uint16_t a, uint16_t b);
08 uint16_t getMax(uint16_t a, uint16_t b);
```

- Rückgabewert: 0: OK; 1: Fehler
  - 0: OK
  - 1: Fehler
- Vorgehen:
  - Header-Datei mit Modulschnittstelle (und Include-Guards)
  - Implementierung des Moduls (Sichtbarkeit beachten)
  - Testen des Moduls im Hauptprogramm (inkl. Fehlerfälle)