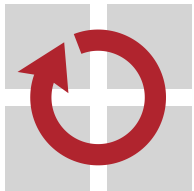
Übungen zu Grundlagen der systemnahen Programmierung in C (GSPiC) im Wintersemester 2017/18

2017-11-21

Bernhard Heinloth

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



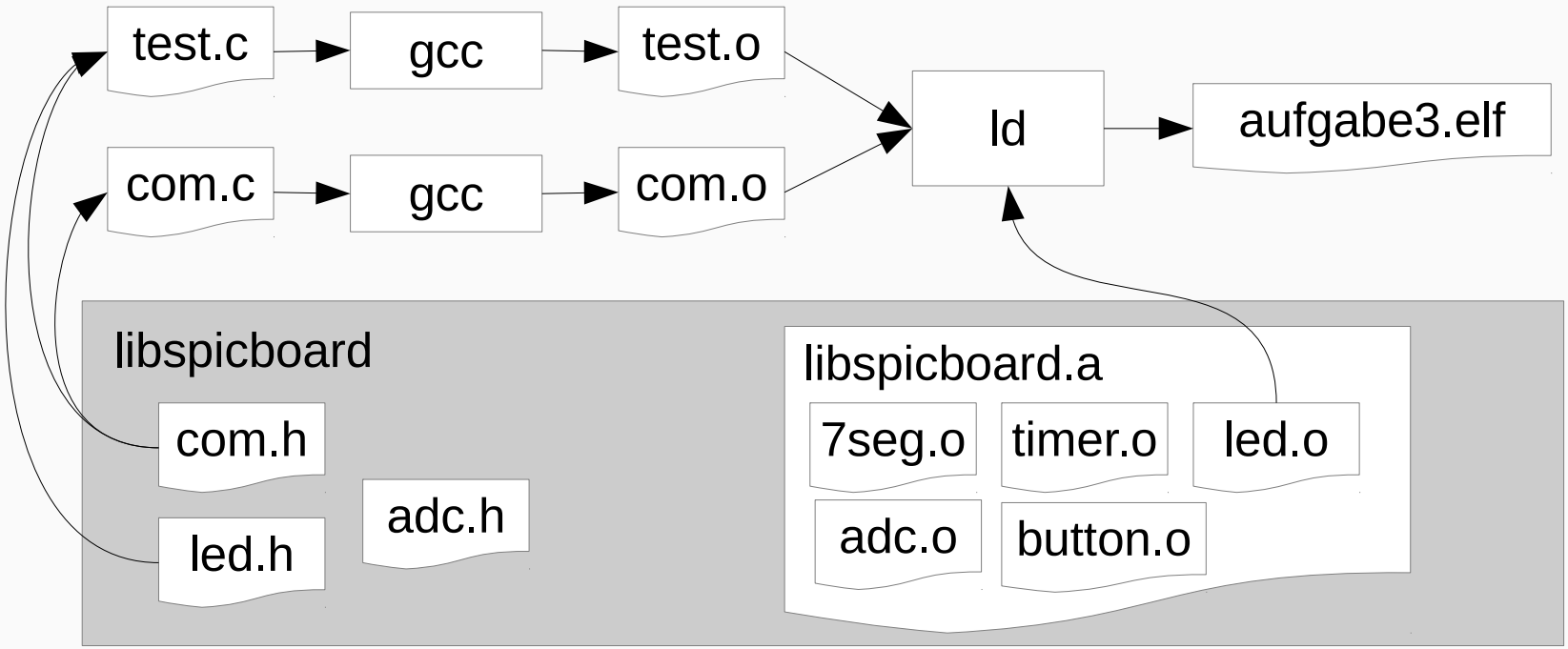
Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Module

Ablauf vom Quellcode zum laufenden Programm



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

Schnittstellenbeschreibung

- 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 '_'
  - Inhalt nur einbinden, wenn das Makro noch nicht definiert ist
- Vorsicht: flacher Namensraum ⇝ möglichst eindeutige Namen

Schnittstellenbeschreibung

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

```
01 #ifndef COM_H
02 #define COM_H
03 /* fixed-width Datentypen einbinden (im Header verwendet) */
04 #include <stdint.h>
05
06 /* Datentypen */
07 typedef enum {
08     ERROR_NO_STOP_BIT, ERROR_PARITY,
09     ERROR_BUFFER_FULL, ERROR_INVALID_POINTER
10 } COM_ERROR_STATUS;
11
12 /* Funktionen */
13 void sb_com_sendByte(uint8_t data);
14 ...
15 #endif //COM_H
```

Initialisierung eines Moduls

- 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

Initialisierung eines Moduls

- `initDone` ist initial `0`
- wird nach der Initialisierung auf 1 gesetzt

⇝ Initialisierung wird nur ein mal durchgeführt

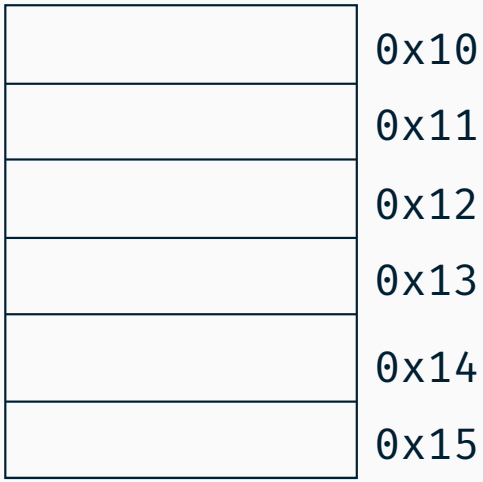
```
01 static void init(void){
02     static uint8_t initDone = 0;
03     if (initDone == 0) {
04       initDone = 1;
05       ...
06     }
07 }
08
09 void mod_func(void) {
10     init();
11     ...
```

Zeiger & Felder

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

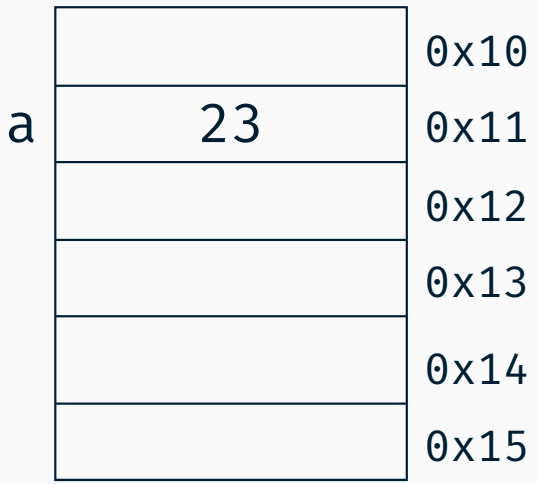


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

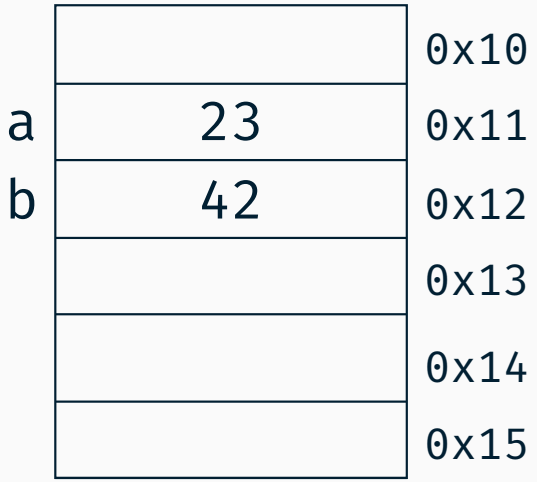


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

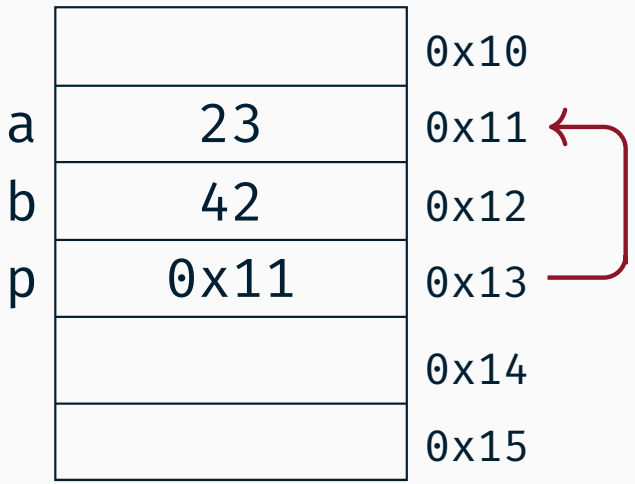


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

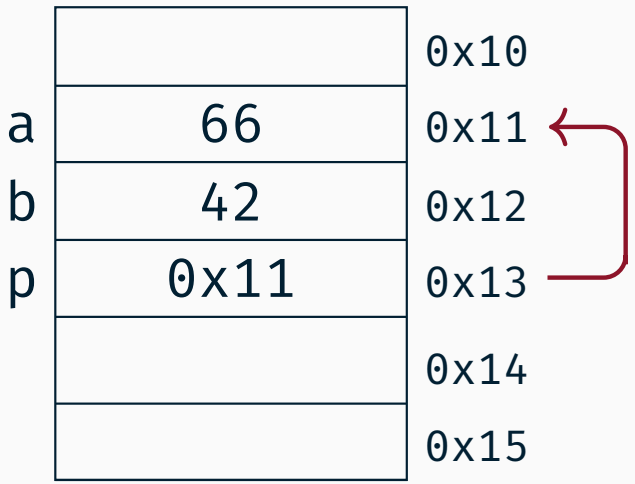


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

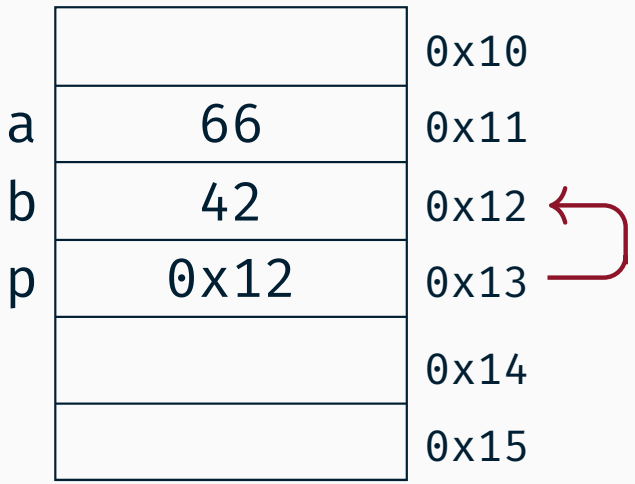


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

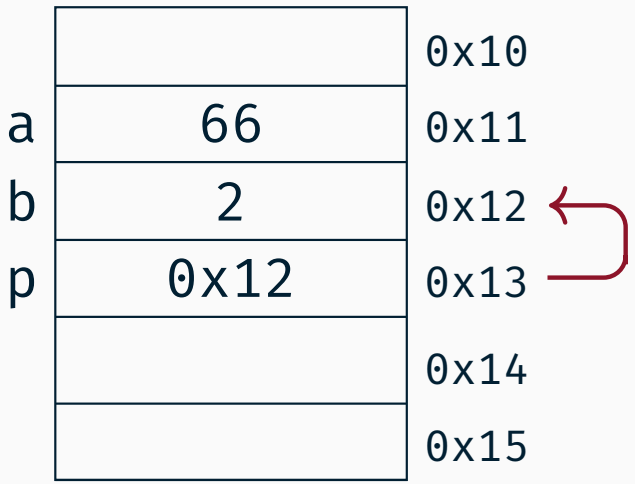


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

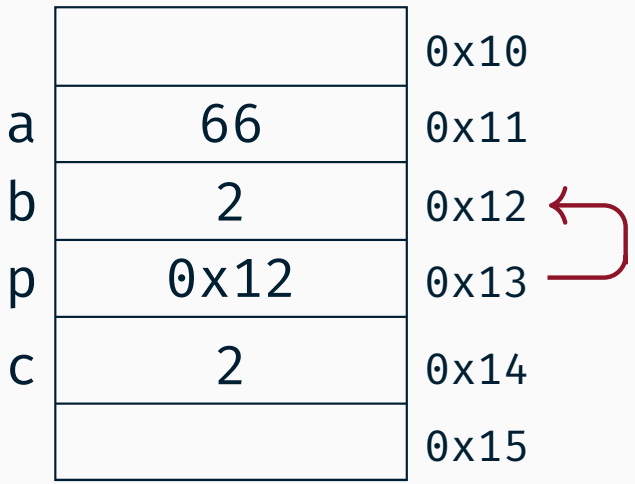


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p = 2;
07 uint8_t c = *p;
```

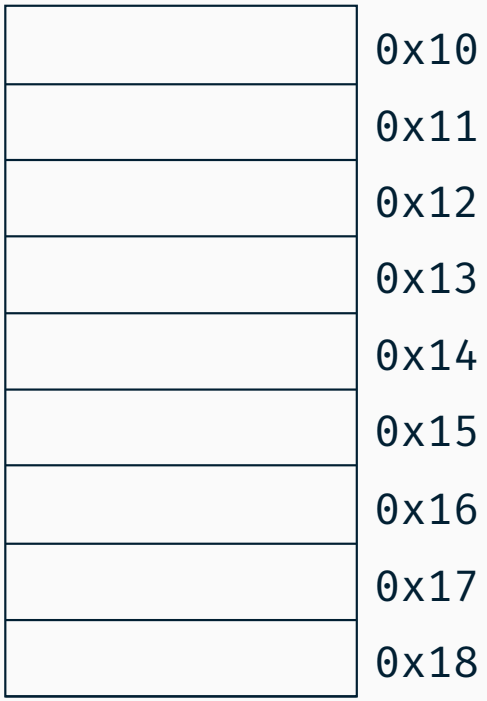


- Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

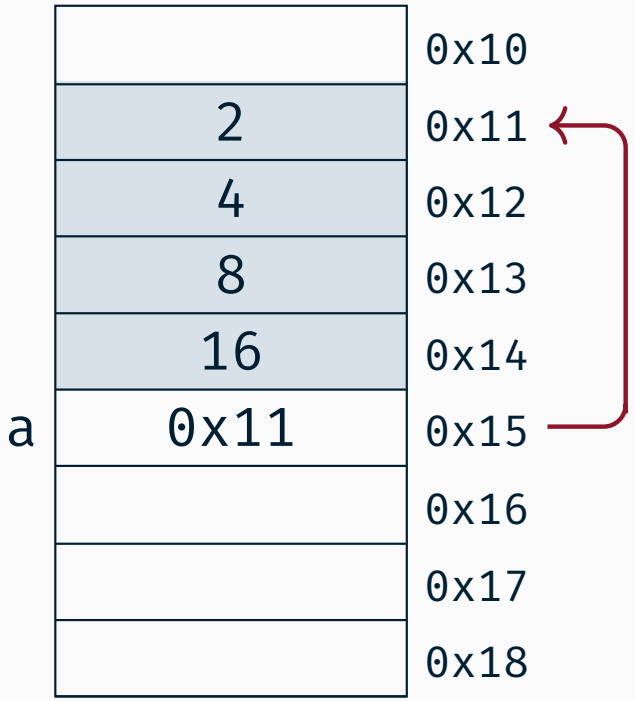
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

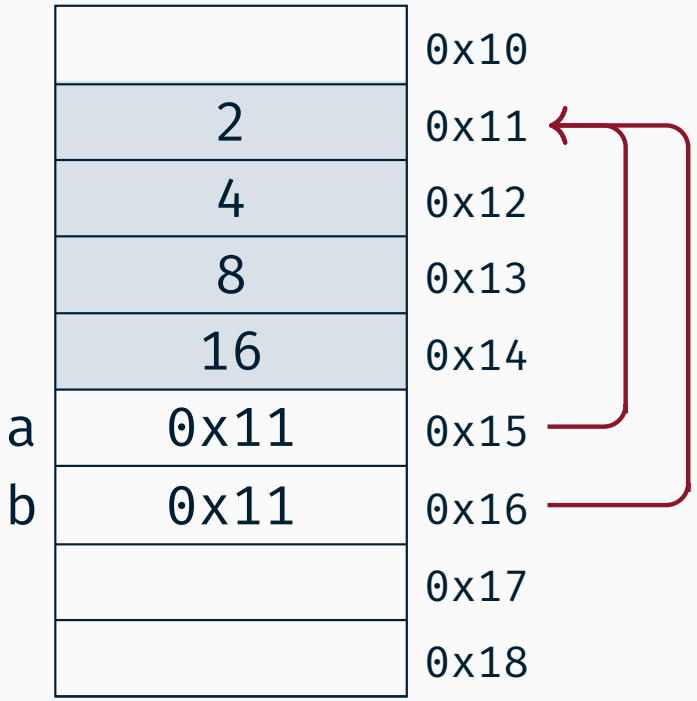
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

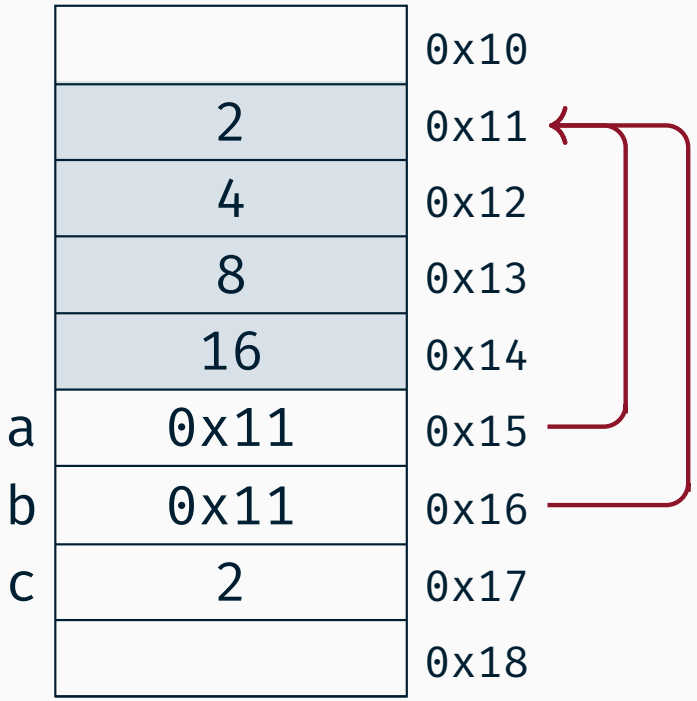
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

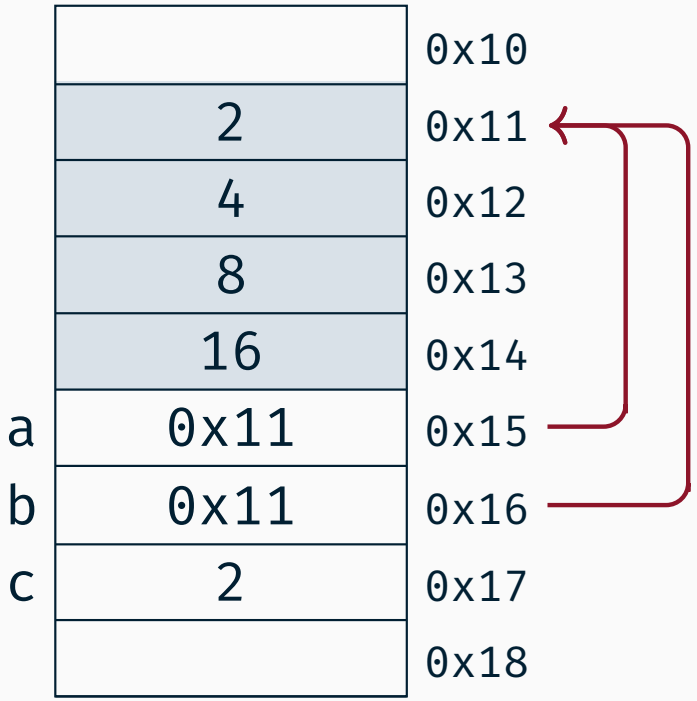
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

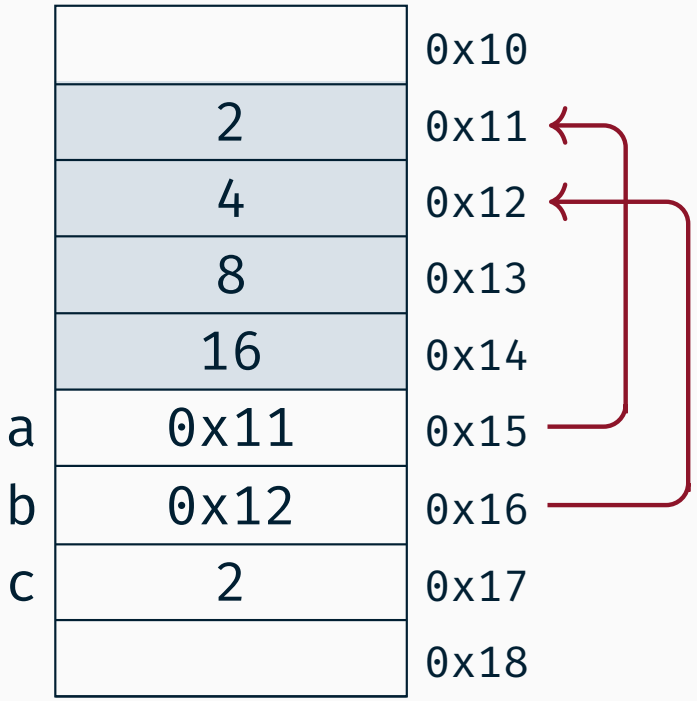
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

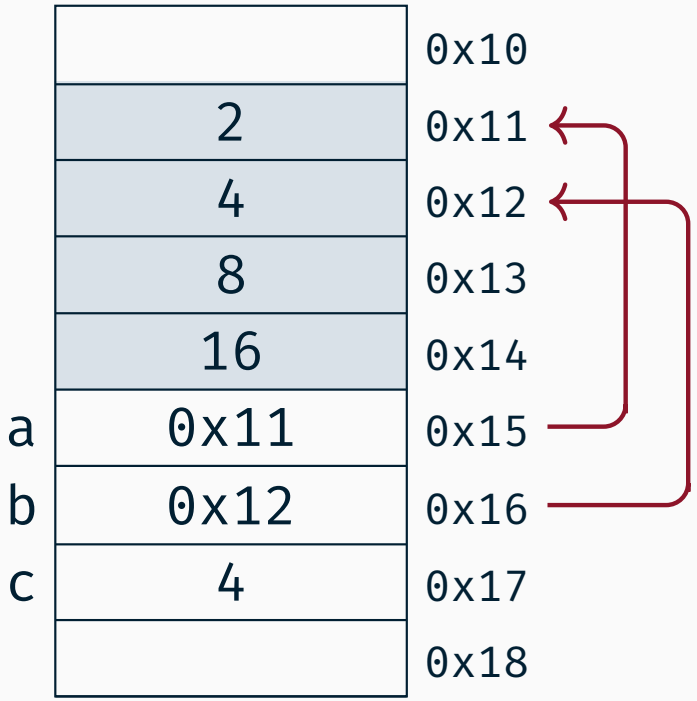
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

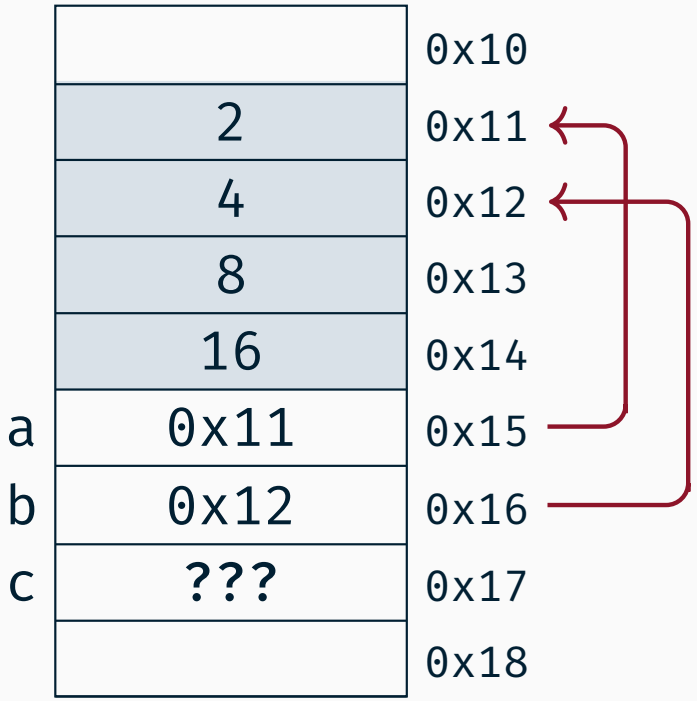
```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7];
```



Wiederholung: Felder

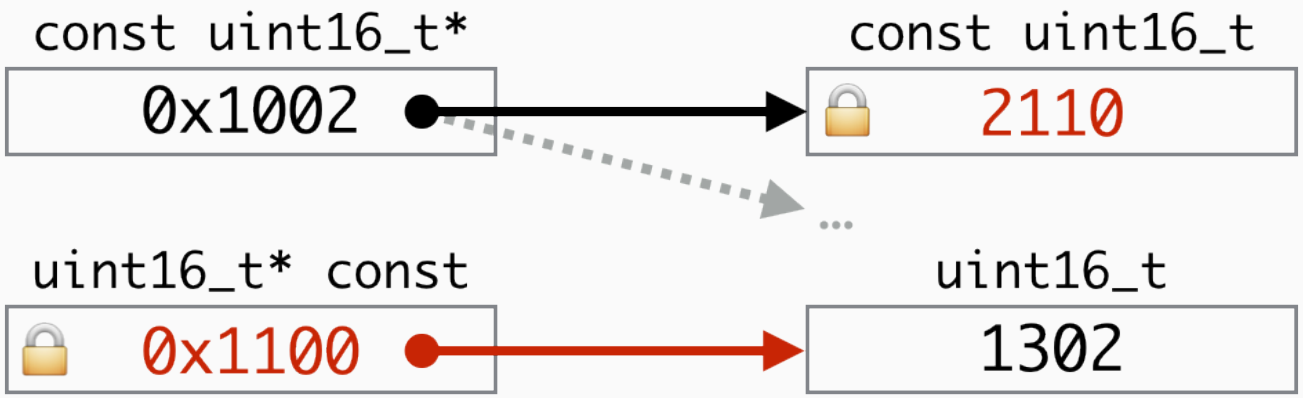
- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelle Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
01 uint8_t a[] = {2,4,8,16};
02 uint8_t *b = a;
03 uint8_t c = a[0];
04 c = *b;
05 b = b+1;
06 c = *b;
07 c = a[7]; // ϟϟϟ
```

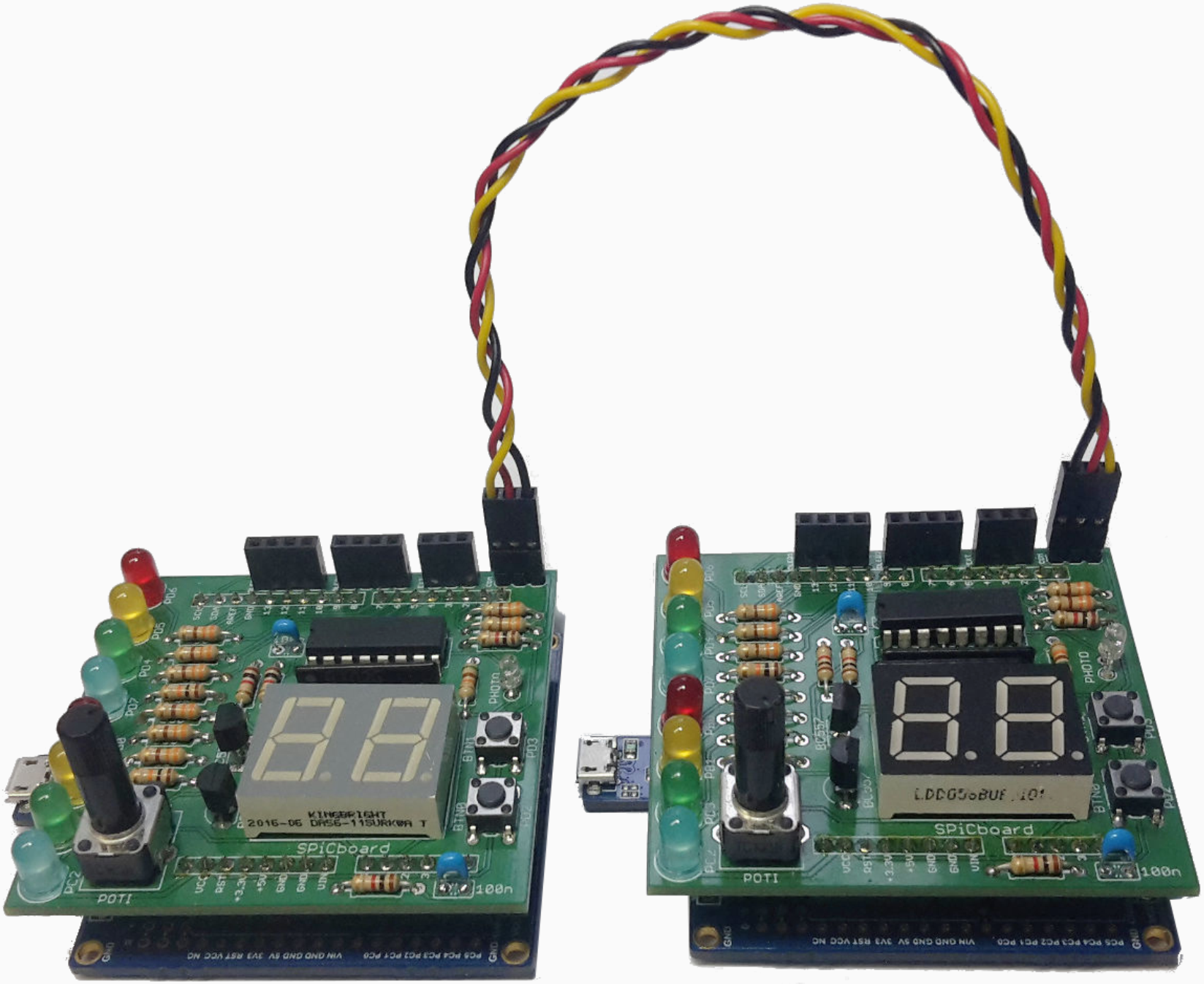


const uint8_t* vs. uint8_t* const

- `const uint8_t*`
  - ein Pointer auf einen `uint8_t`-**Wert**, der konstant ist
  - Wert nicht über den Pointer veränderbar
- `uint8_t* const`
  - ein **konstanter Pointer** auf einen (beliebigen) `uint8_t`-Wert
  - Pointer darf nicht mehr auf eine andere Speicheradresse zeigen



Kommunikation


SPiCboard
POTI
100n
SPiCboard
POTI
100n

SPiCboard verbinden

PD1 Ausgang (TX) wird mit RX des Kommunikationspartners verbunden

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

GND wird mit GND verbunden

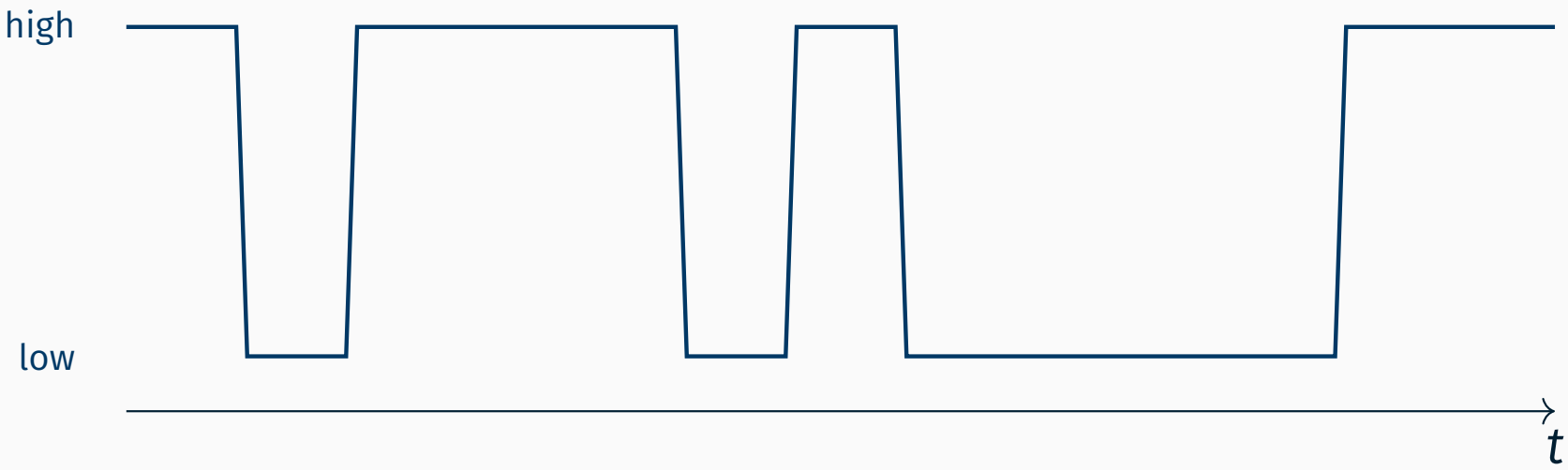
Voraussetzungen für den Datenaustausch

- gleiches Protokoll
  - wir nehmen `8-E-1`

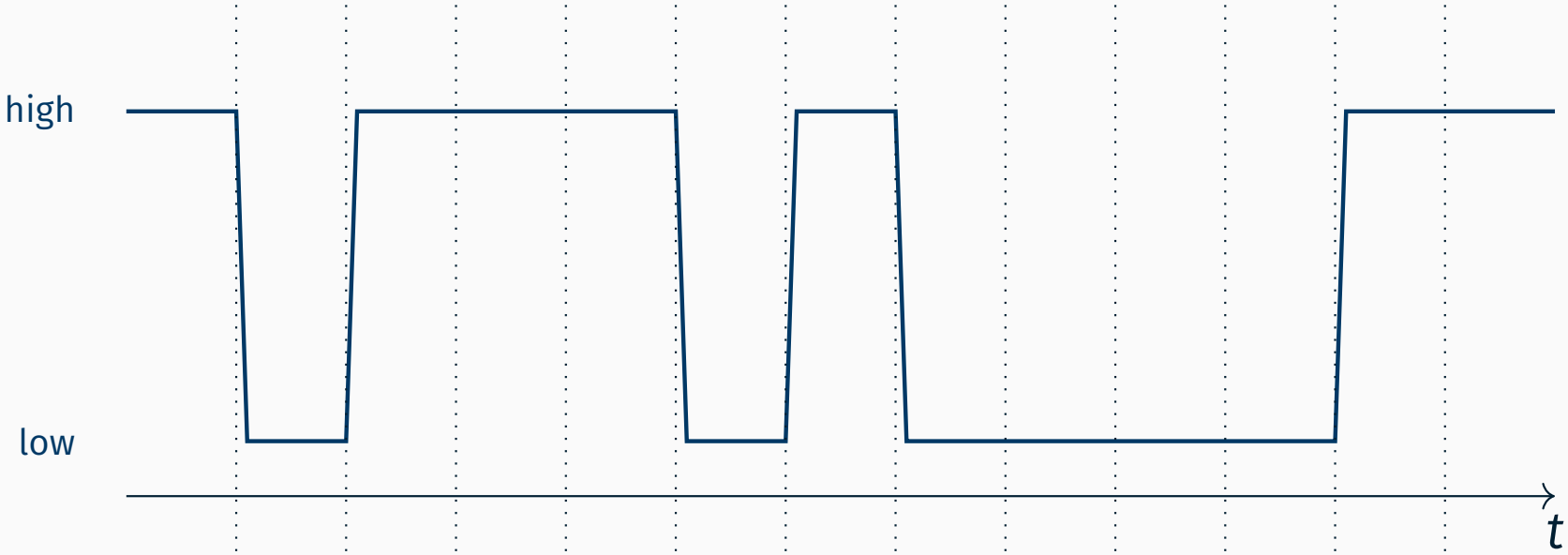
    `8` Anzahl der Datenbits
    `E` gerades (*even*) Paritätsbit
    `1` Stopbit
  - sowie (implizit) ein Startbit
  - das Datenbit mit dem niedrigsten Stellenwert wird zuerst übertragen (aufsteigend)
  - hoher Pegel entspricht einer logischen 1, niedriger Pegel einer `0`
- gleiche Geschwindigkeit, bei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

  (1 Baud entspricht ein Symbol pro Sekunde)

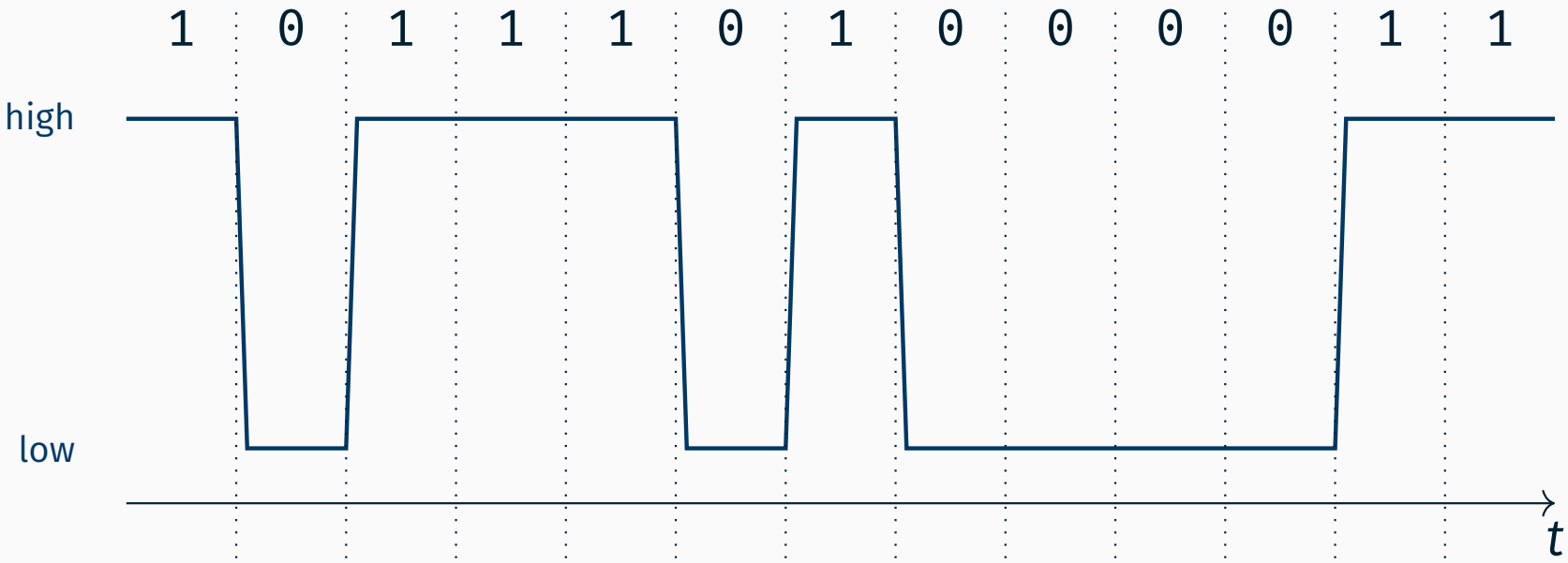
Beispiel



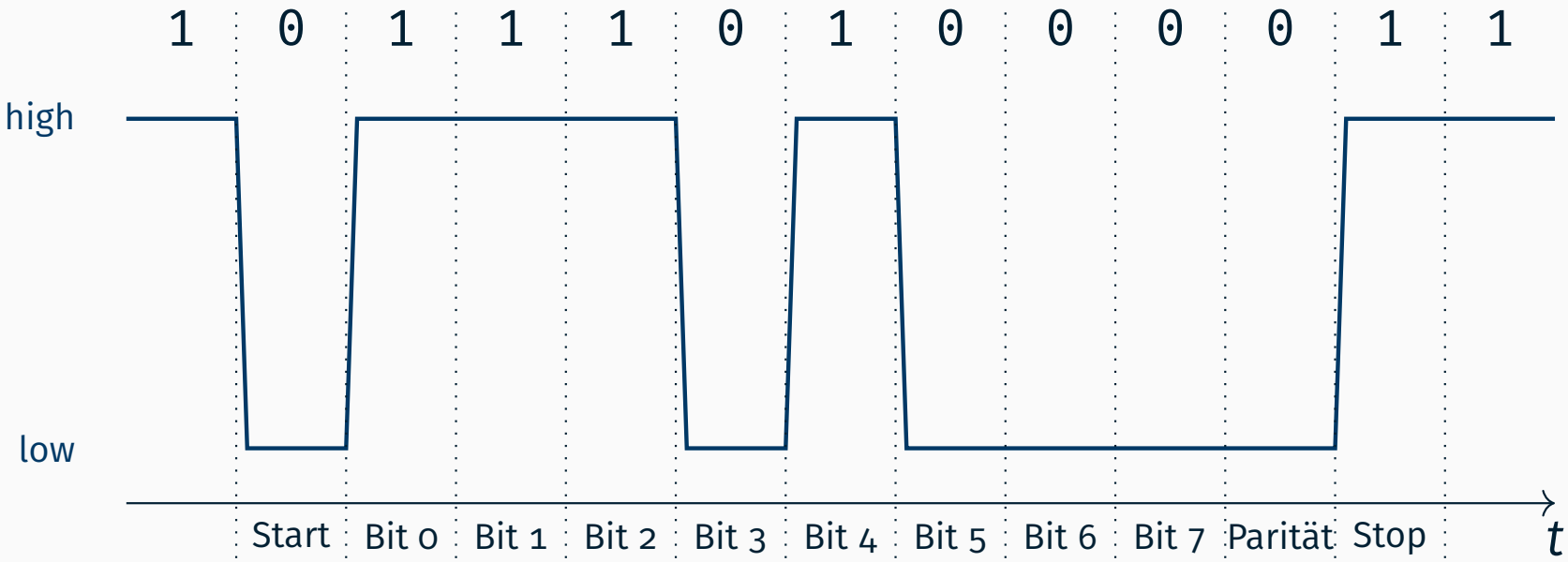
Beispiel



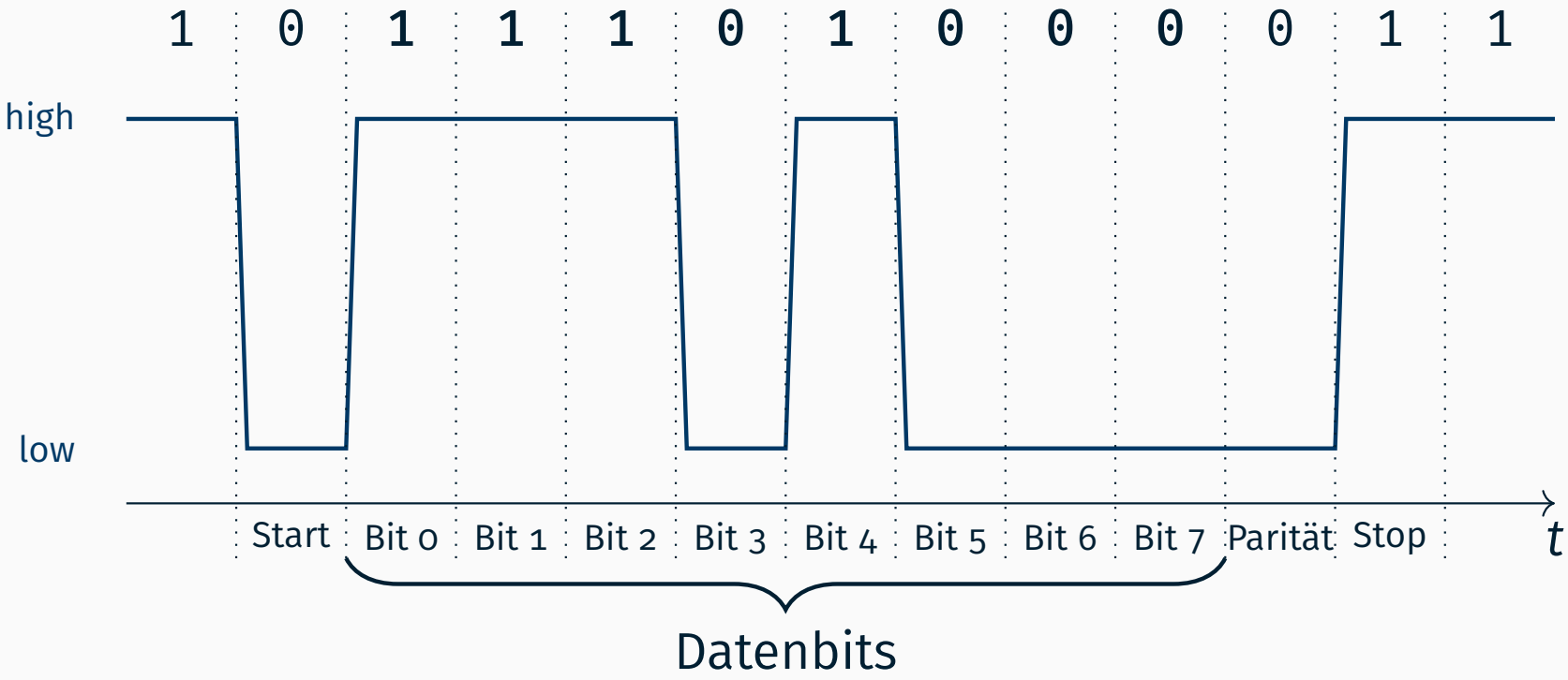
Beispiel



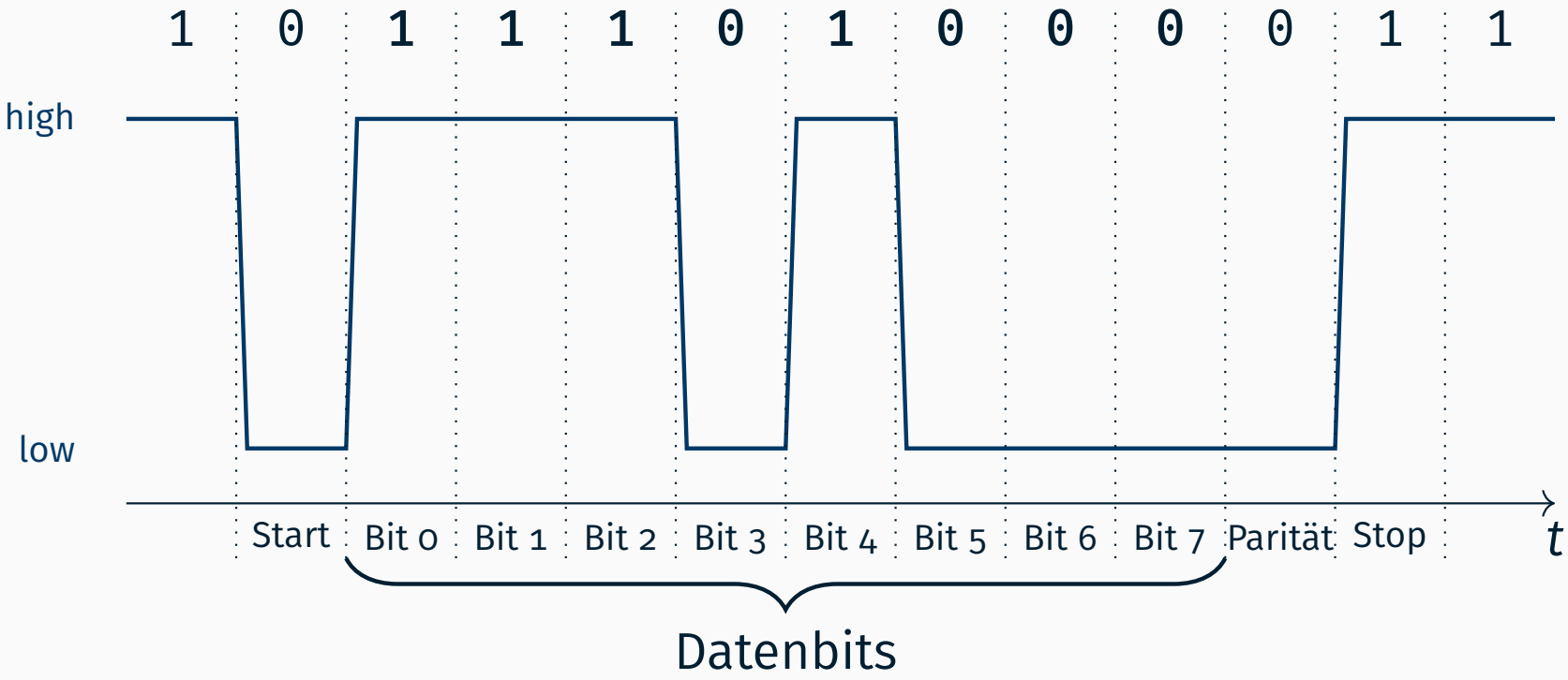
Beispiel


1 0 1 1 1 0 1 0 0 0 0 1 1
high
low
Start
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Parität
Stop
t

Beispiel

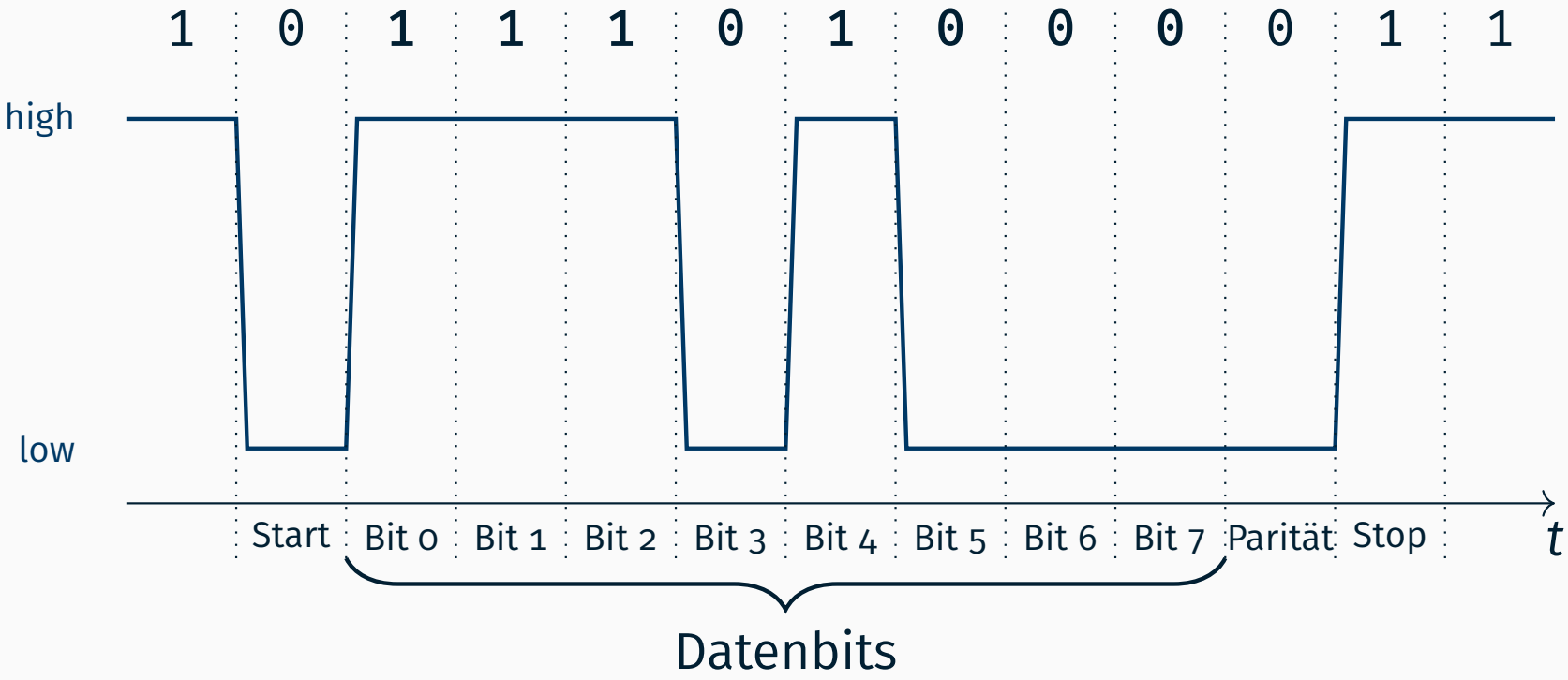

1 0 1 1 1 0 1 0 0 0 0 1 1
high
low
Start
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Parität
Stop
t
Datenbits

Beispiel



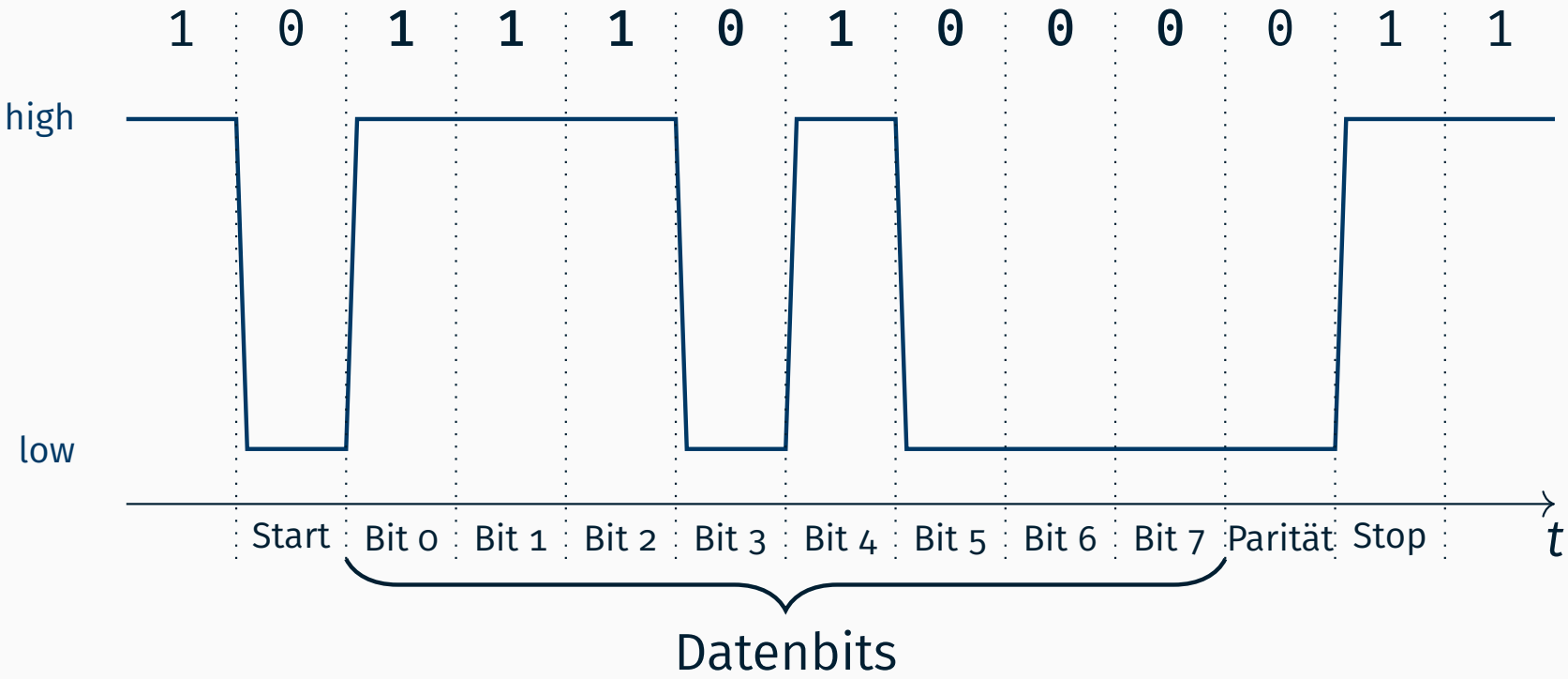
- Empfangenes Byte (rückwärts gelesen): $00010111_2 = 0x17 = 23$

Beispiel



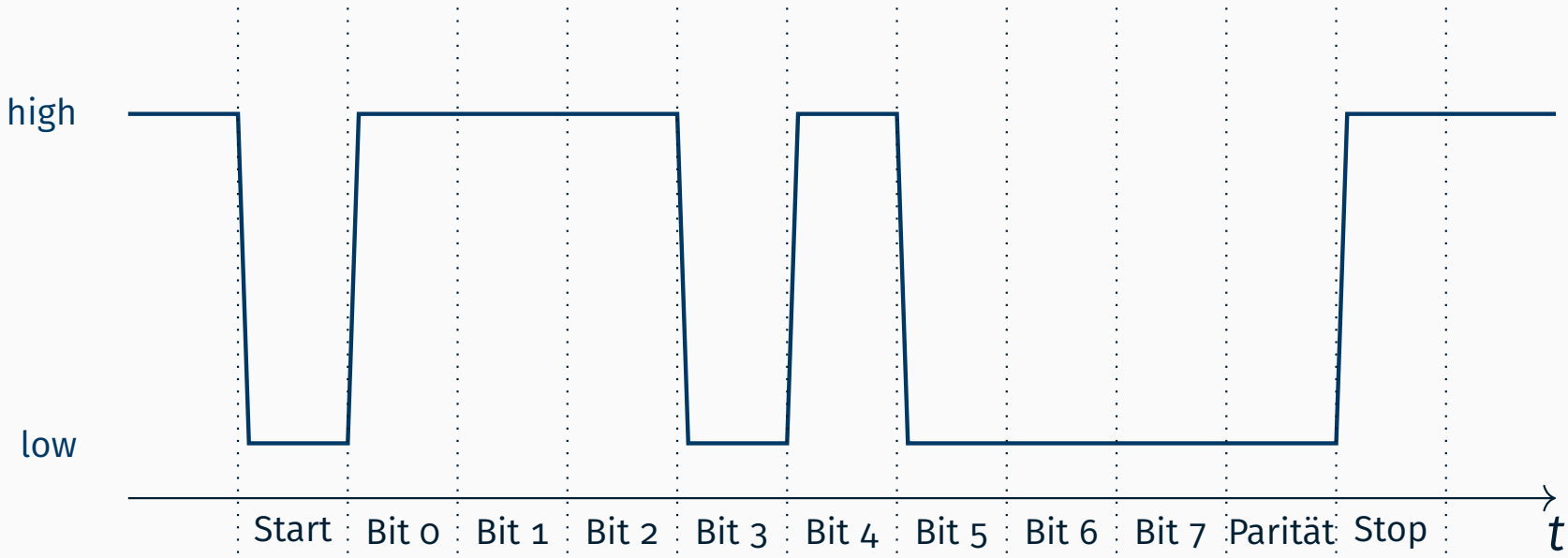
- Empfangenes Byte (rückwärts gelesen): $00010111_2 = 0x17 = 23$
- Parität 0 (da Datenbits vier 1er ⇒gerade Anzahl)

Beispiel



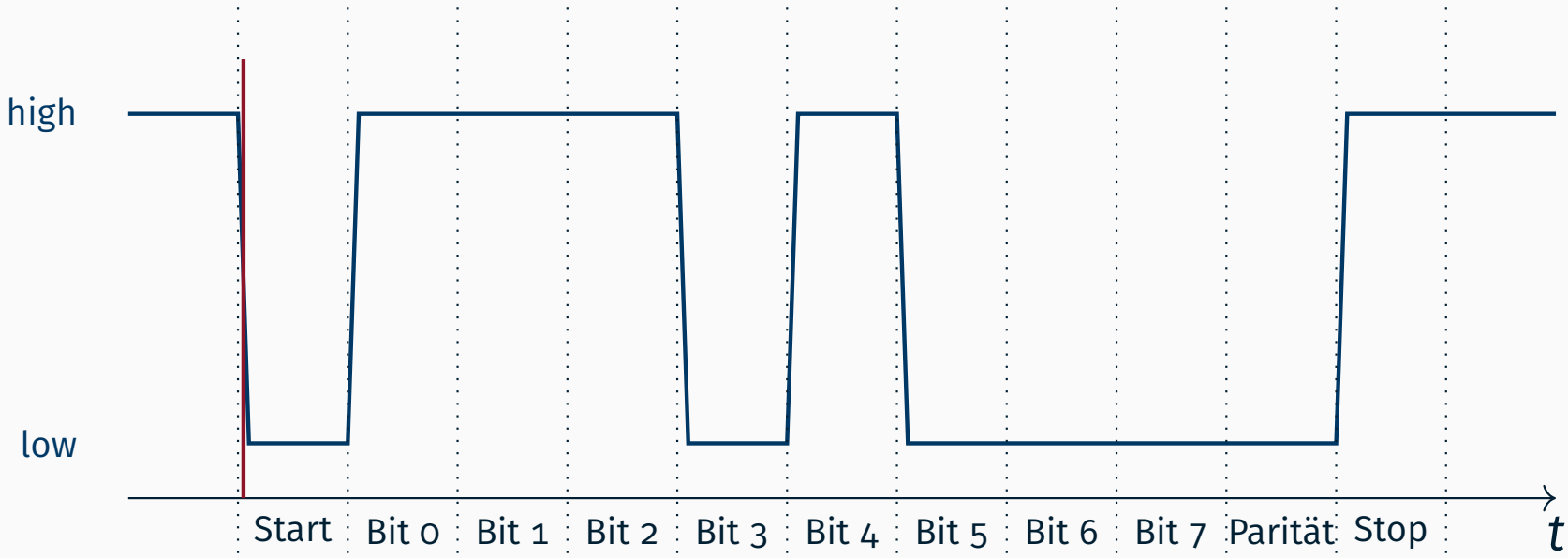
- Empfangenes Byte (rückwärts gelesen): $00010111_2 = 0x17 = 23$
- Parität 0 (da Datenbits vier 1er ⇒gerade Anzahl)
- Startbit immer 0 und Stopbit immer 1

Abtastung



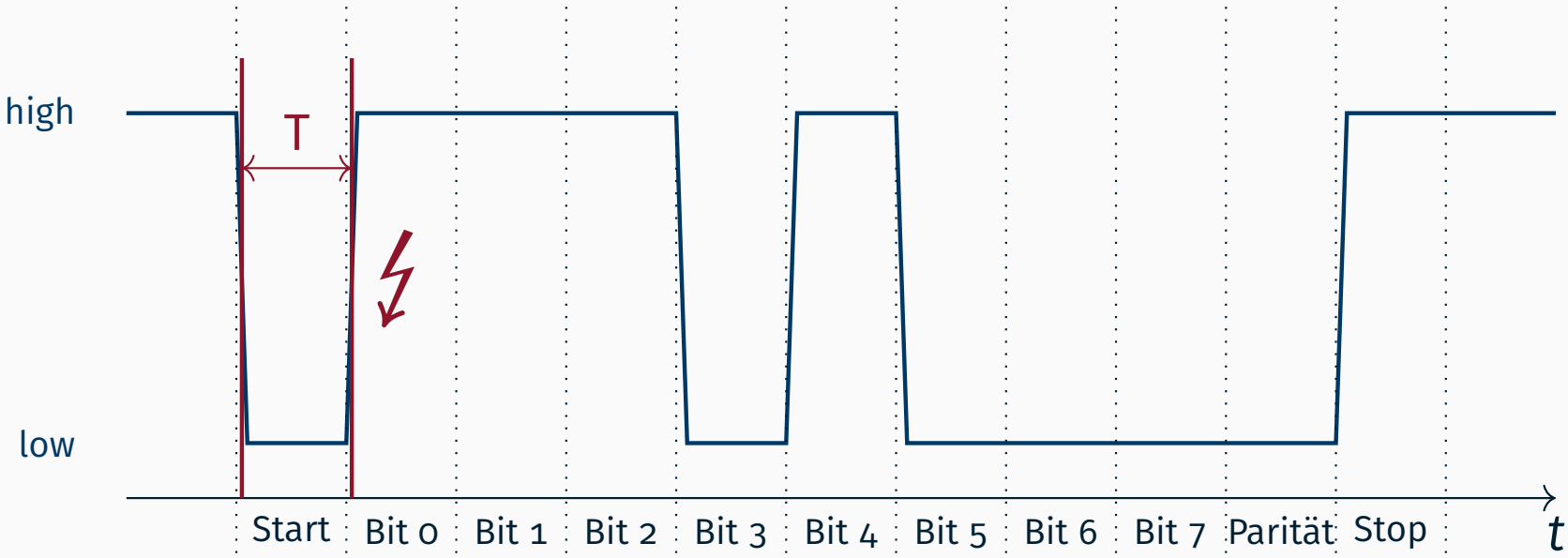
- Die *Fenstergröße* **T** ist uns bekannt

Abtastung



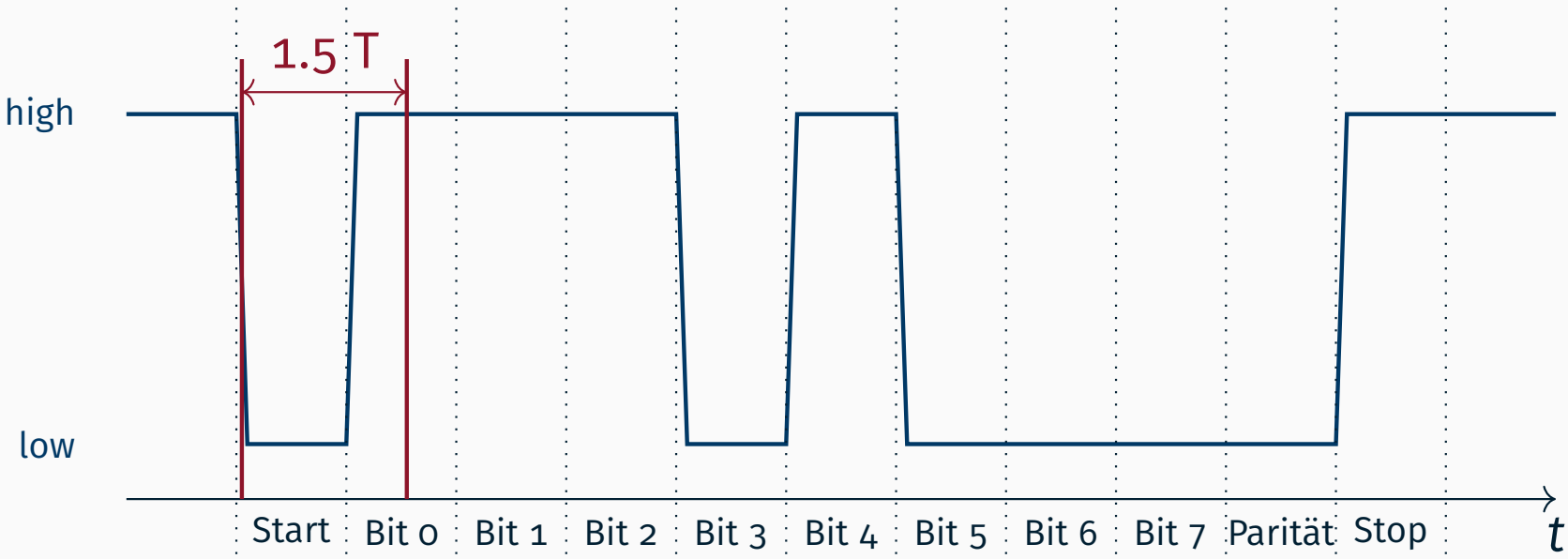
- Die *Fenstergröße* **T** ist uns bekannt
- Anfang ist einen Pegelwechsel von hoch auf niedrig

Abtastung



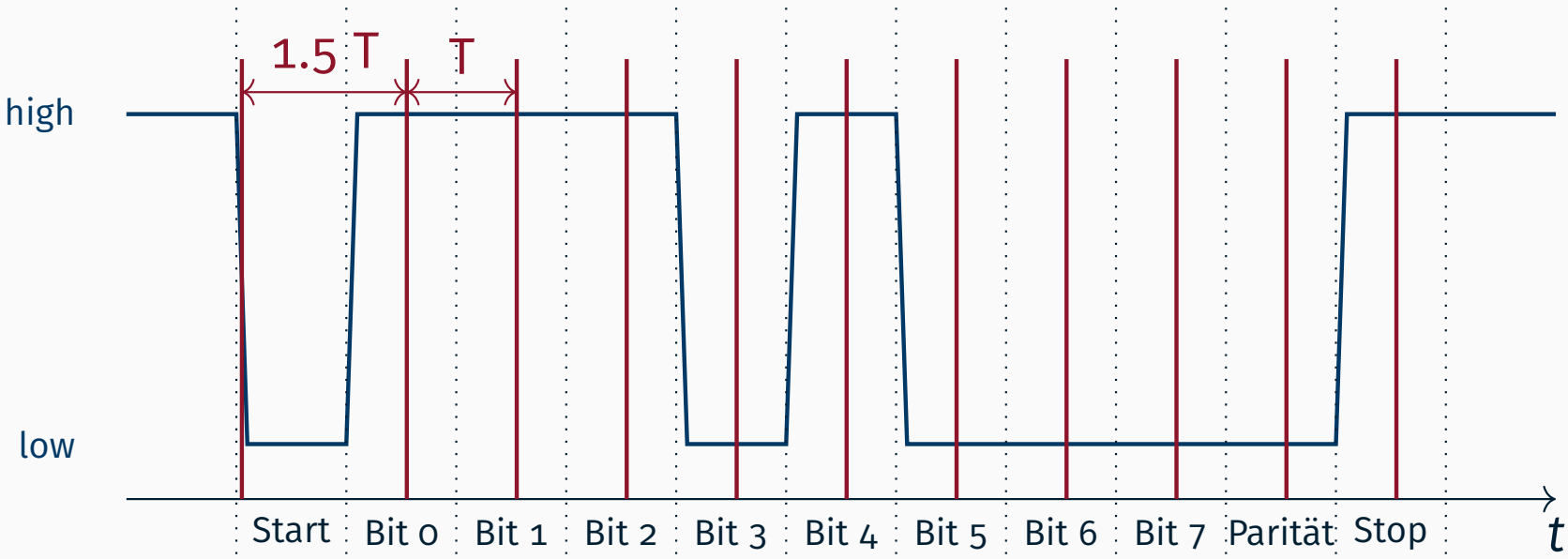
- Die *Fenstergröße* **T** ist uns bekannt
- Anfang ist einen Pegelwechsel von hoch auf niedrig

Abtastung



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

Abtastung



- 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

Aufgabe: Eigenes Kommunikationsmodule

COM-Modul – Aufgabe

- COM-Modul der SPiCboard-Bibliothek selbst implementieren
  - Gleiches Verhalten wie das Original
  - Beschreibung in der Doku auf der Webseite:
    `http://www4.cs.fau.de/Lehre/WS17/V_SPIC/Uebung/doc`

```
01 void sb_com_sendByte(uint8_t data);
02 void sb_com_sendDoubleByte(uint16_t data);
03 void sb_com_sendArray(uint8_t *data, size_t len);
04
05 uint8_t sb_com_receiveByte(uint8_t data);
06 uint8_t sb_com_receiveDoubleByte(uint16_t data);
07 uint8_t sb_com_receiveArray(uint8_t *data, size_t len);
08
09 uint8_t sb_com_byteReadyForReceive();
```

COM-Modul – Aufgabe

- Hilfsfunktionen zum Setzen/Auslesen der Register gegeben
  - Setzen des Pegels für das Senderegister TX
  - Auslesen des Pegels für das Empfangsregister RX
- Hilfsfunktionen gehören nicht zur Schnittstelle
  - ⇝ Sichtbarkeit mit `static` einschränken

```
01 static inline void sb_com_setTx(uint8_t bit) {
02     if (bit)
03         PORTD |= (1 << PD1);
04     else
05         PORTD &= ~(1 << PD1);
06 }
07
08 static inline uint8_t sb_com_getRx() {
09     return (PIND & (1 << PD0)) != 0;
10 }
```

COM-Modul – Aufgabe

- Funktionen der Schnittstelle müssen überall sichtbar sein
  - ⇝ Sichtbarkeit nicht einschränken
- Kompatibilität von Schnittstelle und Implementierung durch Inkludieren des Headers gewährleistet

```
01 uint8_t sb_com_byteReadyForReceive(){
02     sb_com_init();
03     return sb_com_getRx() != 0;
04 }
```

COM-Modul – Aufgabe

- Implementierung nur einmal machen
- Funktionen auf allgemeinste Funktion abbilden
  - `sb_com_sendByte()` auf Feld der Größe 1 abbilden
  - `sb_com_sendDoubleByte()` auf Feld der Größe 2 abbilden

```
01 void sb_com_sendByte(uint8_t data) {
02     sb_com_sendArray(&data, 1);
03 }
04
05 void sb_com_sendDoubleByte(uint16_t data) {
06     sb_com_sendArray((uint8_t *)(&data), 2);
07 }
08
09 void sb_com_sendArray(uint8_t * data, size_t len){
10     // Implement
11 }
```

COM-Modul – Aufgabe

- Analog für Empfangsfunktionen
- Funktionen auf allgemeinste Funktion abbilden
  - `sb_com_receiveByte()` auf Feld der Größe 1 abbilden
  - `sb_com_receiveDoubleByte()` auf Feld der Größe 2 abbilden

```
01 uint8_t sb_com_receiveByte(uint8_t *data) {
02     return sb_com_receiveArray(data, 1);
03 }
04
05 uint8_t sb_com_receiveDoubleByte(uint16_t *data) {
06     return sb_com_receiveArray((uint8_t *) data, 2);
07 }
08
09 uint8_t sb_com_receiveArray(uint8_t *data, size_t len) {
10     // Implement
11 }
```

COM-Modul – Aufgabe

- 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

COM-Modul – Aufgabe

- Aufgabe ist in drei Teile unterteilt
- Teilaufgabe b: Sendefunktionalität
  - Unabhängig vom Empfangen implementieren
  - Implementieren der Funktion `sb_com_sendArray()`
  - Ablauf für jedes 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`

COM-Modul – Aufgabe

- Aufgabe ist in drei Teile unterteilt
- Teilaufgabe c: Empfangsfunktionalität
  - Unabhängig vom Senden implementieren
  - Implementieren der Funktion `sb_com_receiveArray()`
  - Ablauf für jedes 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`

COM-Modul – Aufgabe

- 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.