Informatikwerkstatt, Foliensatz 5
Treiber mit Schrittfunktionen

G. Kemnitz

Institut für Informatik, TU Clausthal (IW5)

14. November 2016

Inhalt:

Wiederholung
EA-Hintergrundprozesse
Treiber LCD-Monitor
Treiber PC-Kommunikation
Treiber Sonar-Sensor
Test der Treiber
Ablaufkontrolle mit LA
Aufgaben

Interaktive Übungen:

1. Test Treiber der mit Schrittfunktion (F5-comsf/comsf)
2. Test der Zeitabläufe mit dem Logikanalysator (USBLOGI)

Wiederholung

Wiederholungsaufgabe 5.1: USART

Entwickeln Sie ein Programm für den Mikrorechner mit Schaltern an Port A, LEDs an Port J, dass den Header »com_pc.h«

```
#ifndef COM_PC_H_
#define COM_PC_H_

#include <avr/io.h>

 void com_pc_init();              //Init. USART2
 uint8_t getByte();               //Byte empfangen
 void sendByte(uint8_t dat); //Byte versenden

#endif /* COM_PC_H_ */
```

verwendet und

- in einer Endlosschleife auf ein Bytes vom PC wartet,
- dessen Wert auf die LEDs ausgibt und
- den Wert plus den Schalterwert modulo-128[1] zurücksendet.

[1] Divisionsrest durch 128, bzw. nur die Bits 0 bis 6.

Lösung

```
#include "com_pc.h"
uint8_t b, c, d;
int main(){
 DDRA = 0;           // Schaltereingabe
 DDRJ = 0xFF;        // LED-Ausgabe
 com_pc_init()       // USART2 initialisieren
 while(1){           // Endlosschleife
  b = getByte();     // Byte empfangen
  PORTJ = b;         // Ausgabe an die LEDs
  c = (b+PINA);      // Bytewert + Schalterwert
  d = c &0x7F;       // mod128 (Bit 7 löschen)
  sendByte(d);       // Ergebnis zurücksenden
 }                   //
}
```

Wiederholungsaufgabe 5.2: Modultest

1. Wie wurden bei uns bisher Module programmiert?
2. Wie wurden bisher Module mit privaten Daten programmiert? Wie wurde dabei das Geheimnisprinzip »keine Zugriffsmöglichkeit fremder Programmteile auf private Daten anderer Programme« sichergestellt?
3. Was ist ein Testrahmen?
4. Welche Möglichkeiten für die Programmierung von Testrahmen hatten wir bisher besprochen?

Lösung

1. Module waren bisher Codefolgen und Unterprogramme.
2. In einer gesonderten c-Datei mit den privaten Daten als globale Variablen und Schnittstellenweitergabe in Headern.
   Geheimnisprinzip: Keine Weitergabe der Adressen privater Daten an Programmteile in anderen c-Dateien.
3. Testrahmen sind Hauptprogramme, die Eingaben bereitstellen, das zu testende Programmmodul ausführen und die Ergebnisse ausgeben oder kontrollieren.
4. Möglichkeiten für die Programmierung von Testrahmen:
   - Test mit Schaltereingaben und LED-Ausgaben in einer Endlosschleife.
   - Schleife über ein Feld von als Konstanten definierten Testeingaben und Abarbeitung im Simulator im Schrittbetrieb oder mit Unterbrechungspunkten.
   - Test vom PC über eine serielle Schnittstelle.

Wiederholungsaufgabe 5.3: Logikanalysator

1. Wie funktioniert ein Logikanalysator und wozu wird er benutzt?
2. Wie können mit unserem Logikanalysator Programmabläufe untersucht werden?
3. Welche vorbereitenden Schritte erfordert das?

Lösung

1. Logikanalysatoren zeichnen binäre Signalfolgen auf.
2. Indem in das Programm Ausgaben an einen Testport einprogrammiert werden.
3. Vorbereitene Schritte:
   - Einprogrammierung der Testausgaben.
   - Bei einem Modultest Programmieren des Testrahmens.
   - Anschluss des Logikanalysators.
   - Schreiben der xml-Konfigurationsdatei (aufzuzeichnende Signal, Aufzeichnungstakt, Triggerbedingung und Pre-Trigger).
   - LA- und Programmstart in der richtigen Reihenfolge.

EA-Hintergrundprozesse

EA-Hintergrundprozesse

- Eine Byteübertragung dauert ≈ 6.000 Maschinenbefehle.
- Die Hardware-Schnittstellen können zeitgleich senden und empfangen.
- Der Prozessor kann, während ein Task[2] auf Abschluss einer EA-Operation wartet, einen anderen Task, der bereit ist, weiter abarbeiten.

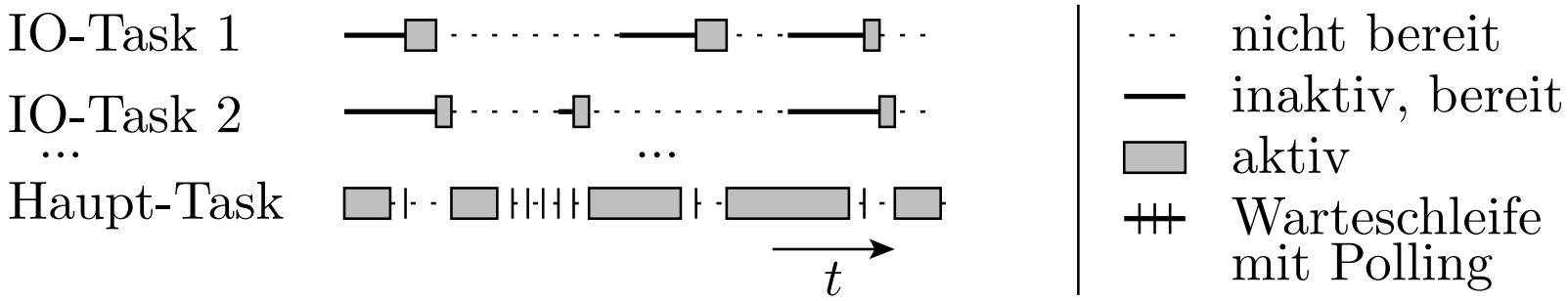
Geplante Software-Architektur mit nebenläufigen[3] EA-Tasks:

- Treiber für alle EA-Tasks, die nur bei Bereitschaft abgearbeitet werden (Tasks ohne Warteschleifen).
- Haupt-Task mit blockierungsfreien EA-Operationen und einer zentralen Warteschleife, in der zyklisch alle EA-Tasks auf Bereitschaft abgefragt werden.

[2] Task: Abzuarbeitende Aufgabe.

[3] Nebenläufig: quasi zeitgleich und ohne zeitliche Ausrichtung.

Geplantes Task-Scheduling



- Wenn der Haupt-Task keine Arbeit hat, fragt er reihum die EA-Tasks ab, ob sie bereit sind. Wenn einer bereit ist, Abarbeitung bis zum Start der nächsten Ein- oder Ausgabe.
- Falls kein Task bereit ist, wiederholt der Haupt-Task die Abfrage zyklisch.
- Nach Abarbeitung aller bereiten EA-Tasks hat der Haupt-Task möglicherweise wieder Daten für seine Fortsetzung.
- Voraussetzung für die rechtzeitige Bedienung aller EA-Geräte sind hinreichend kurze Task-Abarbeitungszeiten.

Treiber für EA-Tasks

Ein EA-Task ist ein Programmobjekt mit

- privaten Daten,
- einer Initialisierungsfunktion für private Daten und zugeordnete Hardware zum Aufruf beim Programmstart,
- Funktionen zum blockierungfreien Lesen und Schreiben von Ein- oder Ausgabedaten zum Aufruf durch andere Programmteile
- und einer blockierungsfrei arbeitenden Schrittfunktion, die vom Haupt-Task zyklisch aufzurufen ist.

Blockierungsfrei: Ohne Warteschleifen. Wenn keine Bereitschaft, Rücksprung mit Rückgabewert »kein Erfolg« (keine Daten übergeben oder übernommen).

Programmgerüst einer Schrittfunktion

- Wenn nicht bereit, sofort Rücksprung,
- sonst, Task-Schritt abarbeiten,
- Zustand aktualisieren und Rücksprung.

```
//private Variablen des Treibers inkl. Zustand
void Schrittfunktion_IO_Task_x(){
  if (!<bereit>) return;
  switch (<Zustand>){ //Falluntersch. nach Zustand
   case <Zustand 1>:  //für den ersten Zustand
     ...              //Anweisungen incl. Zustand
     return;          //weiterschalten und Rückspr.
   case <Zustand 2>:  //für den zweiten Zustand
     ...
  }
}
```

- Beim Programmstart aufzurufende Initialisierungsfunktion:

```
void init_IO_Task_x(...){
  //Initialisierung der zugeordneten Hardware
  ...
  //Initialisierung privater Daten
  ...
}
```

- Lese- und Schreibfunktionen für die Ein- und Ausgabe, z.B.:

```
uint8_t get_IO_Task_x(<Zeiger Datenziel>){
 if (!<neuen Daten>) //wenn keine neuen Daten
  return 0;          //Rückkehr ohne Erfolg
 //Berechnung + Datenübergabe
 ...
 return ≠0;           //erfolgreiche Rückkehr
}
```

Hauptprogramm mit dem Haupttask

```
int main(){
 //Variablenvereinbarungen für das Hauptprogramm
 ...
 init_Task_x(<Initialisierungswerte>);
 init_Task_y(<Initialisierungswerte>); ...
 while (1){              //Endlosschleife
  Schrittfunktion_Task_x();
  Schrittfunktion_Task_y(); ...
  //Haupt-Task als Automat
  switch (<Zustand_main>){
   case <Zustand 1>://für den ersten Zustand
    if (!<bereit>) continue;
    ...                //Berechnung incl. Folgezustand
     break;            //zurück zum Schleifenbeginn
    case ...:          //für den zweiten Zustand
  }
}
```

Treiber LCD-Monitor

LCD-Monitor und sein Treiber (comsf_lcd.c[4])

- Das LC-Display soll zur Statusausgabe genutzt werden:
  - Programmzustände, Sensorwerte,
  - Eingaben, Fehlerzähler, ...
- Die Schrittfunktion soll zeichenweise zyklisch einen als private Daten gespeicherten Text an das LC-Display senden.
- Die von anderen Programmteilen zu nutzenden Ausgabefunktionen schreiben blockierungsfrei in den privaten Textspeicher.

[4] Projekt P05\F5-comsf\comsf.

Öffnen Sie die Datei »comsf_lcd.h«!

- Fehlerzähler erhöhen. Erhöhung des Zeichens der LCD-Position »pos« in der Reihenfolge 1, 2, .., a, b, ... bis Endwert z:

```
void lcd_incErr(uint8_t pos);
```

- Einzelzeichen auf LCD ausgeben. Ausgabe von »c« auf Position »pos«:

```
void lcd_disp_chr(uint8_t c, uint8_t pos);
```

- Text auf LCD ausgeben. Ausgabe eines Texts der Länge »len« ab Speicheradresse »str« ab LCD-Position »pos«:

```
void lcd_disp_str(uint8_t *str, uint8_t pos,
                                uint8_t len);
```

- Zahl auf LCD ausgeben:

```
void lcd_disp_val(uint32_t val, uint8_t pos,
                                uint8_t len);
```

Öffnen Sie die Datei »comsf_lcd.c«!

Private Daten:

```
uint8_t LCD_dat[32];        //Ausgabetext
uint8_t lcd_idx;            //Indexvariable
```

Schrittfunktion. Diese hat 32 Zustände und verschickt in jedem Zustand ein Zeichen des Ausgabetexts an das LC-Display. Der Zustand ist praktisch der Feldindex:

```
void lcd_step(){
 if (UCSR1A&(1<<UDRE1)){  //wenn Puffer frei
  UDR1 = LCD_dat[lcd_idx];//schicke nächstes
  lcd_idx++;              //Zeichen
  if (lcd_idx>=32) lcd_idx = 0;
 }                        //nach dem letzten
}                         //folgt erstes Zeichen
```

Initialisierungsfunktion

```
void lcd_init(uint8_t *text){
  // Initialisierung Sender USART1
  UCSR1C  = 0b110;              //Übertragungsformat 8N1
  UBRR1   = 51;                 //9600 Baud
  UCSR1B |= (1<<TXEN1);         //Sender ein

  //8 Zeichen LC-Display-Initialisierung
  uint8_t lcd_init_dat[] = "\x1B[0h\x1B[j";
  for (lcd_idx=0; lcd_idx<7; lcd_idx++){
    while (!(UCSR1A&(1<<UDRE1)));
    UDR1 = lcd_init_dat[lcd_idx];
  }

 //Initialisieren des Feldes LCD_dat[]
  for (lcd_idx=0; lcd_idx<32; lcd_idx++)
    LCD_dat[lcd_idx] = text[lcd_idx];
  lcd_idx = 0;                  //Index auf Feldanfang
}
```

Fehlerzähler

Sättigungszähler für Fehlfunktionen während der Programmabarbeitung, z.B. Verlust empfangener Daten, Division durch Null, ... Dazu soll das Zeichen mit der Position »pos« in der Reihenfolge ». 1 2 ... 9 a b ... z« hochzählen und im Zustand »z« (zu viele Fehlfunktionen) verbleiben:

```
void lcd_incErr(uint8_t pos){
  pos &= 0x1F;          //WB auf 0:31 begrenzen
  uint8_t w = LCD_dat[pos];
  if (w>='0' && w<'9') w++;
  else if (w=='9') w = 'a';
  else if (w>='a' && w<'z') w++;
  else if (w=='z');
  else w = '1';
  LCD_dat[pos] = w;
}
```

Einzelzeichen- und Textausgabe

- Ausgabe eines Einzelzeichens »c« (z.B. für einen Automatenzustand) auf Position »pos« des LC-Displays:

```
void lcd_disp_chr(uint8_t c, uint8_t pos){
  LCD_dat[pos & 0x1F] = c;  //Position mod-32
}          // begrenzen und Zeichen schreiben
```

- Ausgabe von »len« Zeichen der Zeichenkette str[] (z.B. ein Eingabetext) ab LCD-Position »pos«:

```
void lcd_disp_str(uint8_t *str, uint8_t pos,
                                uint8_t len){
  while (len){                  //für alle Zeichen
    lcd_disp_chr(*str, pos); //Zeichen schreiben
    str++; pos++; len--;
  }
}
```

Ausgabe von Zahlenwerten (z.B. Sensorwerten)

```
void lcd_disp_val(uint32_t val, uint8_t pos,
                                uint8_t len){
 while (len){
  len--;                    //Ziffernanzahl verringern
  lcd_disp_chr((val%10)+'0',pos+len);//Rest mod 10
  val = val / 10;           //Ziffer und Wert durch 10
 }
 if (val)                   //wenn Stellenzahl zu klein
  lcd_disp_chr('?',pos);//Ersatz 1. Ziff. durch '?'
}
```

- Die Ziffern werden mit der kleinsten Ziffer beginnend, als mod-10-Divisionsrest bestimmt und rückwärts von pos+len-1 beginnend in das Textfeld geschrieben.
- Bei Wertebereichsüberlauf (letztes Divisionsergebnis $\neq 0$) wird die führende Ziffer mit »?« überschrieben.

Zusammenfassung

Der Treiber »comsf_lcd« stellt für das LC-Display an der Sendeleitung von USART1 bereit:

- Initialisierungsfunktion, Schrittfunktion,
- Fehlerzähler erhöhen, Einzelzeichen schreiben,
- Text schreiben, Zahlenwert schreiben.

```
void lcd_init(uint8_t *text);
void lcd_step();
void lcd_incErr(uint8_t pos);
void lcd_disp_chr(uint8_t c, uint8_t pos);
void lcd_disp_str(uint8_t *str, uint8_t pos,
                                uint8_t len);
void lcd_disp_val(uint32_t val, uint8_t pos,
                                uint8_t len);
```

Ein Anwendungsbeispiel mit Testprogramm folgt ab Folie 41.

Treiber PC-Kommunikation

Treiber »comsf_pc« für die PC-Kommunikation

- Erwartet Modul PmodUSBUSART an Port H.
- Initialisierungs-, Schritt- Sende- und Empfangsfunktion:

```
#define COM_PC_SMSG_LEN 4       //Größe Sendepuffer
#define COM_PC_RMSG_LEN 4       //Größe Empfangspuffer

void com_pc_init();                     //Init.-Funktion
void com_pc_step();                     //Schrittfkt.
uint8_t com_pc_get(uint8_t *msg); //Empfangsfkt.
uint8_t com_pc_send(uint8_t *msg);//Sendfkt.
uint8_t com_pc_last_byte(); //letztes empf. Bytes
void com_pc_rmsg_clr();   //löschen unvollst. Nachr.
```

- »Lesen letztes empfangene Byte« dient später zur Übergabe von Sofortanweisungen wie »Auto sofort anhalten«.
- »Löschen unvollständig empfangener Nachrichten« erforderlich bei Übertragungsfehlern, später über Timeout.

Öffnen Sie die Datei »comsf_pc.c«!

Private Daten:

```
uint8_t rmsg[COM_PC_RMSG_LEN];
uint8_t smsg[COM_PC_SMSG_LEN];
uint8_t sidx, ridx,          //Indexvariablen
uint8_t last_byte;           //letztes empf. Byte
```

Initialisierungsfunktion (USART2, PmodUSBUSART an JH) :

```
void com_pc_init(){
 UCSR2C=0b110;                        //8N1
 UBRR2=51;                            //9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2);//Send. + Empf. ein
 //Initialisierung der Puffer
 ridx = 0;                          //Empfangspuffer leer
 sidx = COM_PC_SMSG_LEN;  //Sendepuffer leer
}
```

Schrittfunktion

```
void com_pc_step(){
  if (UCSR2A & (1<<RXC2)){   //wenn Byte empfangen
    last_byte = UDR2;        //Byte lesen
  if (ridx<COM_PC_RMSG_LEN){//wenn noch Platz,
    rmsg[ridx] = last_byte; //in Empfangspuffer
    ridx++;                 //schreiben
    }
  }
  if (UCSR2A & (1<<UDRE2) && sidx<COM_PC_SMSG_LEN){
    UDR2 = smsg[sidx];      //wenn Sendepuffer frei
    sidx++;                 //und Senddaten bereit,
  }                         //diese versenden
}
```

Empfangsfunktionen für das Anwenderprogramm

- Nicht blockierendes Lesen einer Nachricht. Ergebnis ist entweder eine komplette Nachricht aller »COM_PC_RMSG_LEN=4« Bytes oder »keine Nachricht«:

```
uint8_t com_pc_get(uint8_t *msg){
 if (ridx<COM_PC_RMSG_LEN)//wenn Empf.-Puffer nicht
  return 0;                //voll, Rückspr. "erfolglos"
 for (ridx=0; ridx<COM_PC_RMSG_LEN;ridx++)
  msg[ridx] = rmsg[ridx];//sonst Empf.Nachricht kop-
 ridx = 0;                //ieren Empfangspuffer leeren
 return 1;                //Rücksprung "Daten erhalten"
}
```

- Letztes empfangene Byte lesen:

```
uint8_t com_pc_last_byte(){return last_byte;}
```

Sendefunktion (Nachrichten, nicht blockierend)

- Wenn Platz, Nachricht von »COM_PC_RMSG_LEN=4« Bytes in den Sendepuffer kopiert. Rücksprung mit 1 (OK).
- Sonst Rücksprung, ohne ein Byte zu kopieren, mit 0 (Daten nicht übergeben).

```
uint8_t com_pc_send(uint8_t *msg){
 if (sidx<COM_PC_SMSG_LEN)//wenn Sendepuffer nicht
  return 0;               //leer, Rückgabe "erfolglos"
 for (sidx=0; sidx<COM_PC_SMSG_LEN;sidx++)
  smsg[sidx] = msg[sidx];//sonst Nachricht überge-
                         //ben und Zeiger auf Nach-
                         //richtenanfang
 sidx = 0;               //Sendepuffer voll
 return 1;               //Rückspr. "Daten übergeben"
}
```

Ein Anwendungsbeispiel mit Testprogramm folgt ab Folie 41.

Treiber Sonar-Sensor

Öffnen Sie die Datei »comsf_sonar.h«!

Der Treiber erwartet den Sonarsensor gemeinsam mit dem LC Display an JD und stellt bereit:

- Initialisierungsfunktion:

```
void sonar_init();
```

- Schrittfunktion:

```
void sonar_step();
```

- Abholfunktion für den empfangenen Abstandswert. Wenn der Treiber einen neuen Sonarwert empfangen hat, wird dieser auf der Adresse »sptr« gespeichert, der Empfangsautomat neu initialisiert und mit »1« zurückgekehrt. Sonst Rückkehr ohne Werteübergabe mit »0«.:

```
uint8_t sonar_get(uint16_t *sptr);
```

Öffnen Sie die Datei »comsf_sonar.c«!

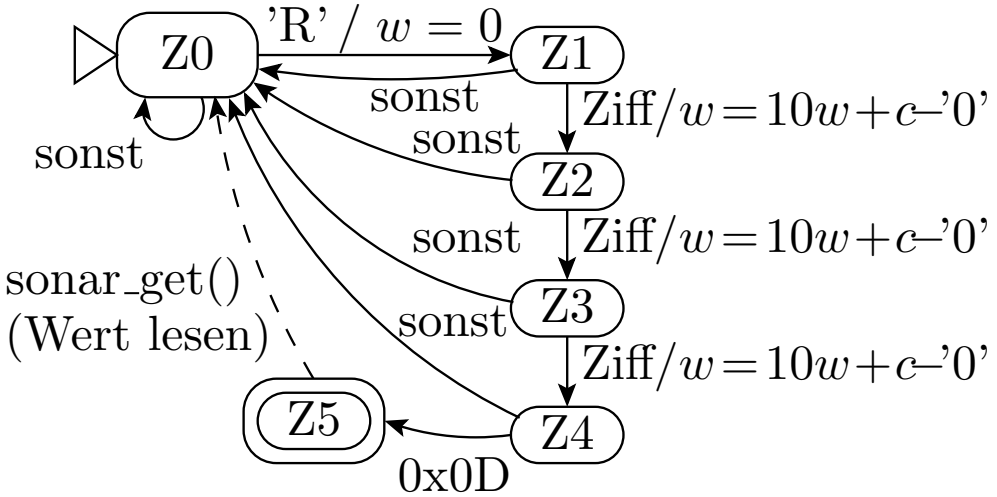
Private Daten sind der Abstandswert und der Treiberzustand:

```
uint16_t snr_val;    //(w) Abstand in Zoll
uint8_t  snr_state; //(z) Zustand Empfangsautomat
```

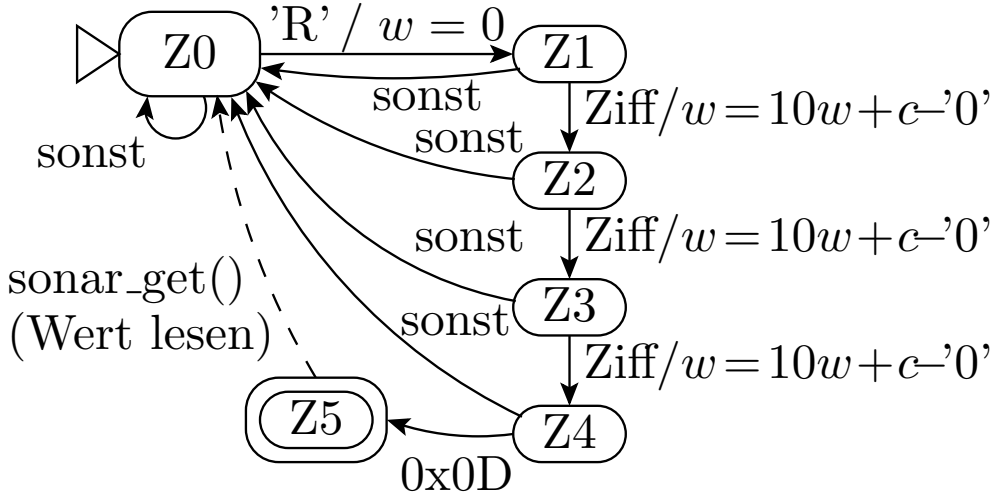
Die Initialisierung schaltet auch den Sensor ein:

```
void sonar_init(){   //Initialisierung USART1
 UCSR1C=0b110;       //Übertragungsformat 8N1
 UBRR1=51;           //9600 Baud
 UCSR1B|=(1<<RXEN1);//Empfänger ein
 DDRD  |= 1<<PD5;    //PD5 Ausgang
 PORTD |= 1<<PD5;    //Sonar einschalten
 snr_state =0;
}
```

Schrittfunktion




```
void sonar_step(){
 if (!(UCSR1A & (1<<RXC1))) //wenn kein neues
  return;                   //neues Zeichen, beenden
 uint8_t dat = UDR1;        //sonst Zeichen lesen
 if (snr_state==0 && dat=='R'){
  snr_state = 1;            //Kante von Z0 nach Z1
  snr_val   = 0;
 }
 ...                        //"Ziff"-Kanten
 ...                        //Kante von Z4 nach Z5
 ...                        //"sonst"-Kanten
}
```




```
                                //"Ziff"-Kanten
  else if (snr_state>0 && snr_state<4 && dat>='0' && dat<
   snr_val = (snr_val*10) + (dat-'0');
   snr_state++;
   return;
  }
   else if                      //Kante von Z4 nach Z5
   (snr_state==4 && dat==0x0D){snr_state = 5;}
  else if (snr_state<5)
   snr_state = 0;               //"sonst"-Kanten
}
```

Abholfunktion für den empfangenen Abstandswert

Bei Abholen des Wertes wird der Zustand rückgesetzt, so dass die Schrittfunktion mit dem Zusammenbau des nächsten Wertes beginnen kann.

```
uint8_t sonar_get(uint16_t *sptr){
  if (snr_state>=5) {//wenn neuer Wert
    *sptr = snr_val; //dann Ausgabe
    snr_state = 0;   //Zustand rücksetzen
    return 1;        //Rücksprung mit "wahr"
  }
  else               //sonst
    return 0;        //Rücksprung mit "falsch"
}
```

Test der Treiber

Überblick über die zu testenden Funktionen

Sende- und Empfangspuffer sind in den Testbeispielen 4 Byte groß:

```
#define COM_PC_SMSG_LEN 4
#define COM_PC_RMSG_LEN 4
```

Zu testen

- sind 3 Initialisierungsfunktionen:

```
void lcd_init(uint8_t *text);
void com_pc_init();
void sonar_init();
```

- 3 Schrittfunktionen:

```
void lcd_step();
void com_pc_step();
void sonar_step();
```

- Die Schreibfunktionen des LCD-Treibers:

```
void lcd_incErr(uint8_t pos);
void lcd_disp_chr(uint8_t c, uint8_t pos);
void lcd_disp_str(uint8_t *str,uint8_t pos,
                                uint8_t len);
void lcd_disp_val(uint32_t val,uint8_t pos,
                                uint8_t len);
```

- Die Empfangs- und Sendefunktionen vom und zum PC:

```
uint8_t com_pc_get(uint8_t *msg);
uint8_t com_pc_last_byte();
uint8_t com_pc_send(uint8_t *msg);
```

- Die Lesefunktion für den Sonarwert:

```
uint8_t sonar_get(uint16_t *sptr);
```

Testrahmen »test_comsf.c« für die drei Treiber

Anzeige auf dem LCD:

- `S:....`: Sonarwert,
- `I:....`: 4-Byte-Empfangsnachricht,
- `L:.`: letztes empfangenes Zeichen,
- `s:....`: Anzahl der Abstandsmessungen durch 50,
- `i:..`: Anzahl der empfangenen Nachrichten,
- `E:...`: Fehlerzähler für Empfangsfehler und zu Testzwecken für emfangene Nachrichten.

Rücksenden nach jedem Nachrichtenempfang:

- Sonarwert (2 Byte) und
- Sonarwertzähler (2 Byte).

Konstantenvereinbarungen für die LCD-Darstellung


```
#define LCD_STR "S:...␣I:....␣L:.s:...␣i:..␣E:..."
//Anzeigepositionen für Zahlen und Texte
#define LCP_SON       2   //Sonarwert "S:..."
#define LCP_RMSG      8   //Eingabedaten "I:...."
#define LCP_LBYTE    15   //letztes empf. Byte
#define LCP_TIME     18   //Anz. Son.-Mess./50
#define LCP_ICT      24   //Anzahl Eingaben
//Anzeigepositionen für Fehlerzähler
#define LCP_COMERR   29   //Kommunikationsfehler
#define LCP_TESTERR  30   //Testfehlerzähler
```

Hauptprogramm

```
int main(void){
 uint8_t mrmsg[COM_PC_RMSG_LEN];//Empfangsnachricht
 uint8_t msmsg[COM_PC_SMSG_LEN];//Sendenachricht
 uint16_t snrval, sct=0, ict=0;
 sonar_init();                  //Treiber initialisieren
 com_pc_init();
 lcd_init((uint8_t*)LCD_STR);
 while(1){                      //Beginn Hauptschleife
  lcd_step();                   //Schrittfunktionen aller
  com_pc_step();                //Treiber aufrufen
  sonar_step();
  ...                           //Bei Eingabeereignissen
  ...                           //auszuführende Aktionen
 }
}
```

Aktionen bei Eingabeereignissen

Bei neuem Sonarwert (ca. 50 mal je s)

- diesen auf das LCD ausgeben,
- Sonarwertzähler erhöhen und
- Sonarzähler/50 als Sekundentakt ausgeben:

```
if (sonar_get(&snrval)){   //wenn neue Sonardaten
  //Sonarwert und Messwertnummer auf LCD schreiben
  lcd_disp_val(snrval, LCP_SON, 3); //Sonarwert
  sct++;      //Sonarwertzähler erhöhen und Wert/50
  lcd_disp_val(sct/50, LCP_TIME, 3);//ausgeben
}
```

Nach Empfang einer neuen PC-Nachricht:

- Empfangene Nachricht auf LCD ausgeben.
- Testfehlerzähler erhöhen und auf LCD ausgeben,
- Sensorwert und Messwertnummer an den PC senden.

- Nach Empfang einer neuen PC-Nachricht ...

```
if (com_pc_get(mrmsg)){  //Wenn neue PC-Nachricht
      //diese in mrmsg übernehmen, auf LCD ausgeben
 lcd_disp_str(mrmsg, LCP_RMSG, COM_PC_SMSG_LEN);
 lcd_incErr(LCP_TESTERR);//Testfehlerzähler erhöhen
    //Eingabenzähler erhöhen und ausgeben
 lcd_disp_val(++ict, LCP_ICT, 2);
 msmsg[0] = snrval >> 8;  //Sensorwert und Messwert-
 msmsg[1] = snrval & 0xFF;//nummer byteweise in den
 msmsg[2] = sct >> 8;     //string "msg" schreiben
 msmsg[3] = sct & 0xFF;
 if (!com_pc_send(msmsg)) //"msmsg" versenden
   lcd_incErr(LCP_COMERR);//wenn senden erfolglos
}                         //Sendfehlerzähler erhöhen
```

- Immer letztes empfangenes Byte auf das LCD schreiben:

```
lcd_disp_chr(com_pc_last_byte(), LCP_LBYTE);
```

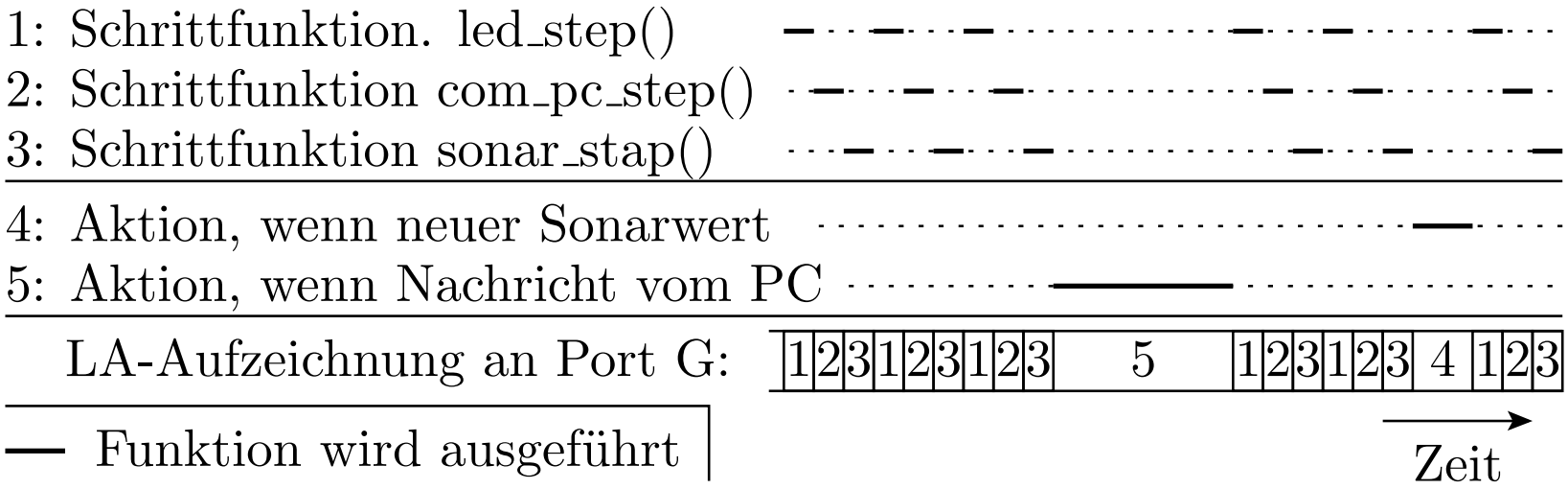
Ausprobieren des Testbeispiels (test_comsf)

Projekt »F5-comsf\comsf« öffnen, übersetzen und Programm starten:

- Der Abstandswert hinter »S:« muss der Abstand vom Ultraschallsensor bis zum nächsten Gegenstand in Zoll sein.
- Der Wert hinter »s:« muss etwa im Sekundentakt weiterzählen und von »999« nach »?00« wechseln. (Bei Überlauf wird aus der ersten Ziffer ein »?«).
- Beim Senden einzelner Zeichen mit dem HTerm muss hinter »L:« immer das letzte empfangene Zeichen und hinter »I:« die letzte empfangene 4-Byte-Nachricht stehen.
- Beim Empfang einer 4-Byte-Nachricht muss sich der Wert hinter »i:« und der zweite Fehlerzäher hinter »E:« erhöhen.
- Der Fehlerzähler muss in der Reihenfolge ». 1 2 ... 9 a b ... z« zählen und im Zustand »z« verbleiben.

Ablaufkontrolle mit LA

Kontrolle der Zeitabläufe mit dem Logikanalysator

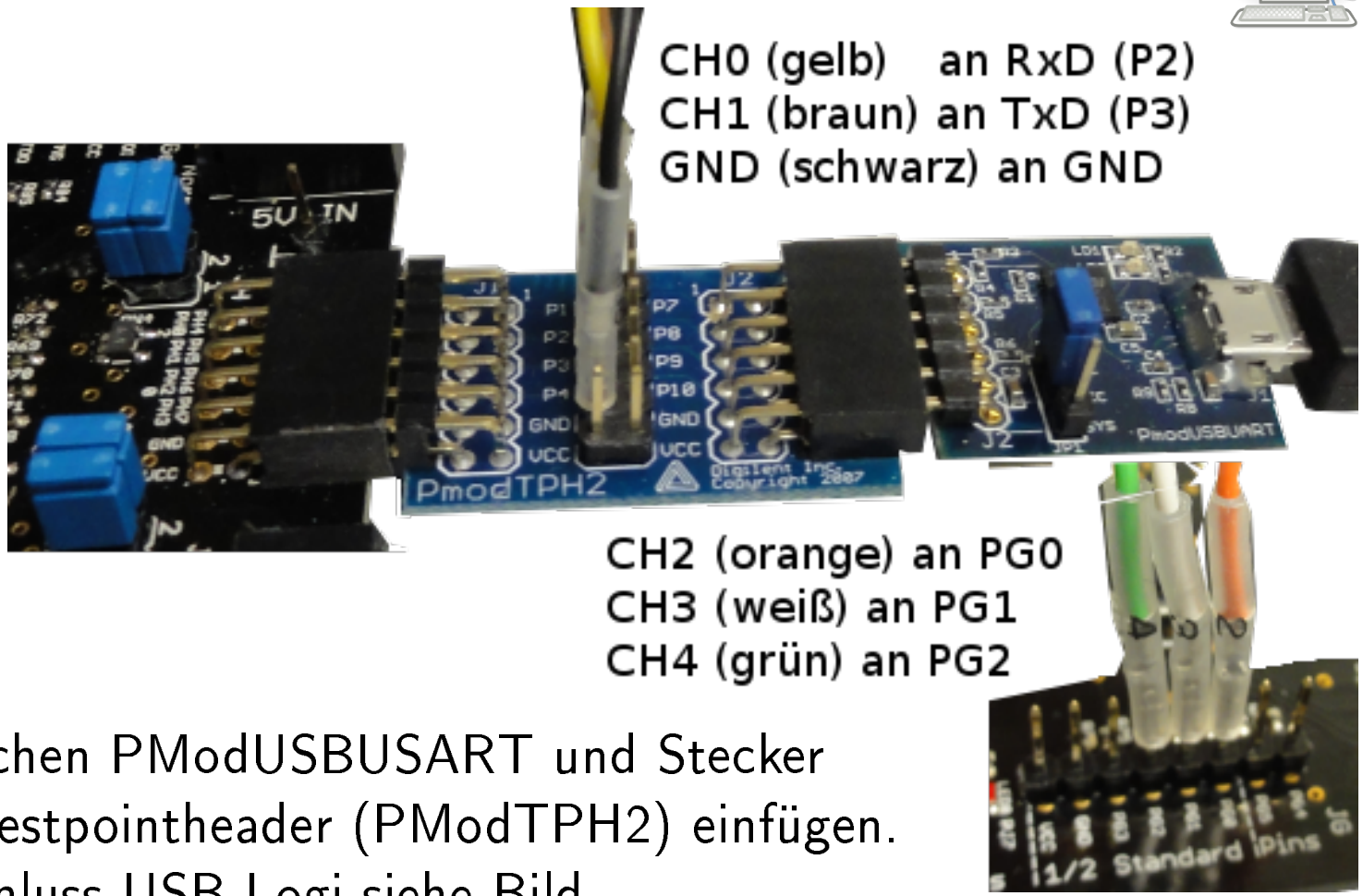


Zur Visualisierung der Dauer und Reihenfolge der Teilaufgaben:

- Ausgabe der Teilaufgabennummer an PG0 bis PG2 und
- Aufzeichnung der Nummern mit dem Logikanalysator.

Zusätzliche Aufzeichnung der seriellen Sende- und Empfangsdaten von und zum PC.

Anschluss des Logikanalysators



- Zwischen PModUSBUSART und Stecker JH Testpointheader (PModTPH2) einfügen.
- Anschluss USB-Logi siehe Bild.
- USB-Logi muss, wenn angesteckt, auch am USB-Kabel stecken. Sonst werden kontaktierte Signale auf null gezogen.

Programmergänzungen in »main()«:

```
int main(void){
 DDRG = 0xFF; // Port G als Ausgang initialisieren
 ...
 while (1){
  PORTG = 1; lcd_step();    //vor jeder Schritt-
  PORTG = 2; com_pc_step();//funktion PORTG++
  PORTG = 3; sonar_step();
  PORTG = 0;
  if (sonar_get(&snrval)){ //wenn neue Sonardaten
   PORTG = 4;              //soll der USB-Logi
   ...                     //0b100 aufzeichnen
  }
  if (com_pc_get(mrmsg)){  //Wenn neue PC-Nachricht
   PORTG = 5;              //soll der USB-Logi
   ...                     //0b101 aufzeichnen
  }
 }
}
```

Konfiguration des USB-Logi (test_comsf.xml)

```
<samplerate>500000</samplerate># 500.000 Werte/s
<pretrigger>6</pretrigger>      # Hälfte Vortrigger-
 <signals>                      # aufzeichnung
  <signal name="Zustand">       # 3-Bit-Bus
    <ch>2</ch>                  # orange
    <ch>3</ch>                  # weiss
    <ch>4</ch>                  # grün
  </signal>
  <signal name="RxD"> <ch>0</ch> </signal> # gelb
  <signal name="TxD"> <ch>1</ch> </signal> # braun
</signals>
<trigger when="A">              # Trigger bei
  <A> <ch when="high">2</ch>    # PortG == 5
      <ch when="low">3</ch>
      <ch when="high">4</ch> </A>
</trigger>
```

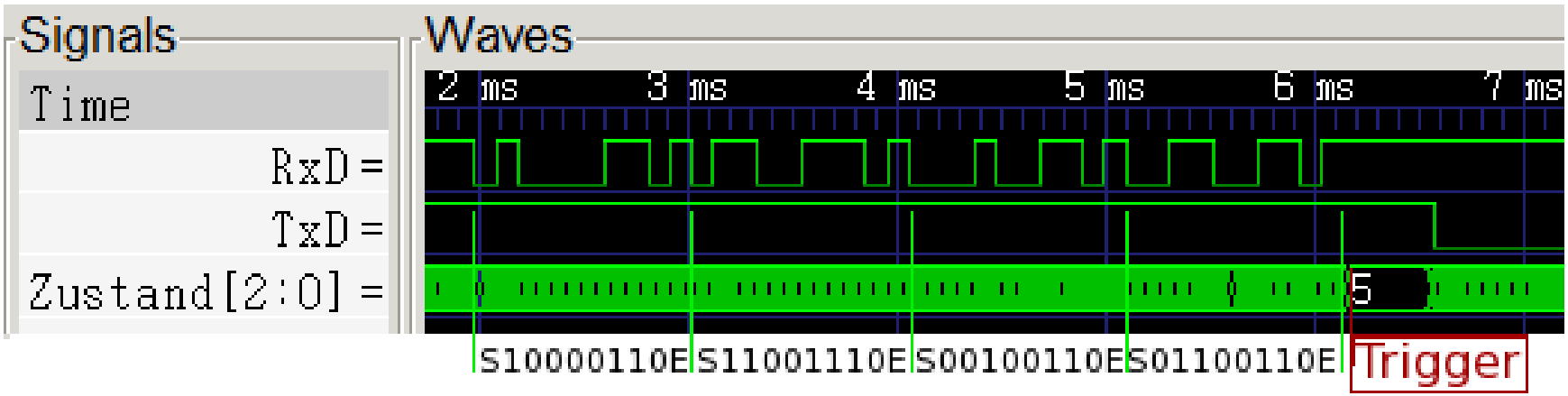
Experiment

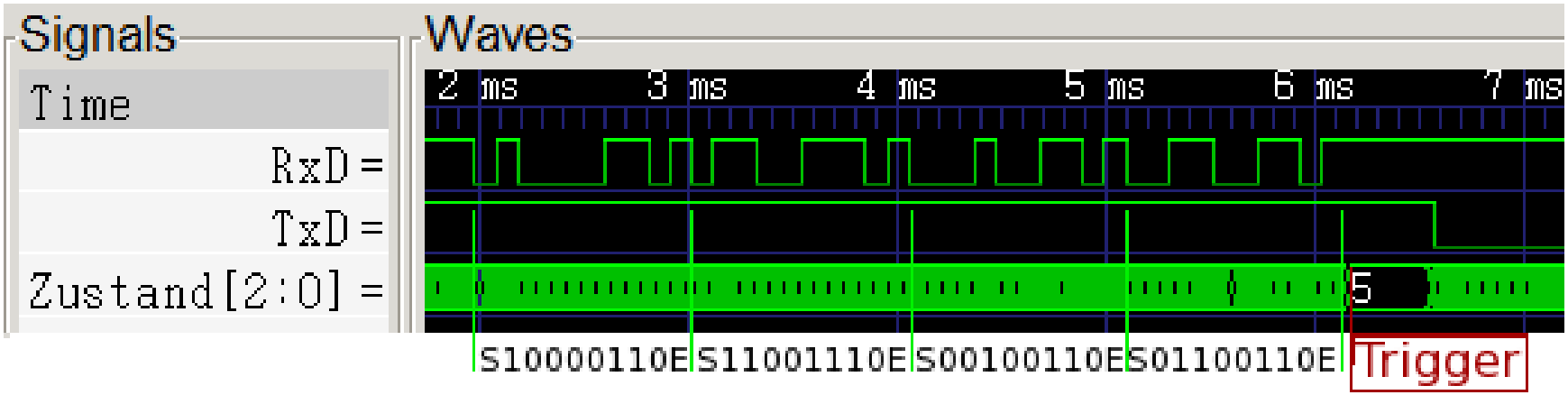
- Programm »...\P05\F5-comsf\test_comsf« läuft.
- Windows-Konsole »cmd« starten. In das Verzeichnis .....\P05\USBLOGI wechseln.
- Logikanalysator starten mit:

```
usb-logi test_comsf.xml test_comsf.sav
```

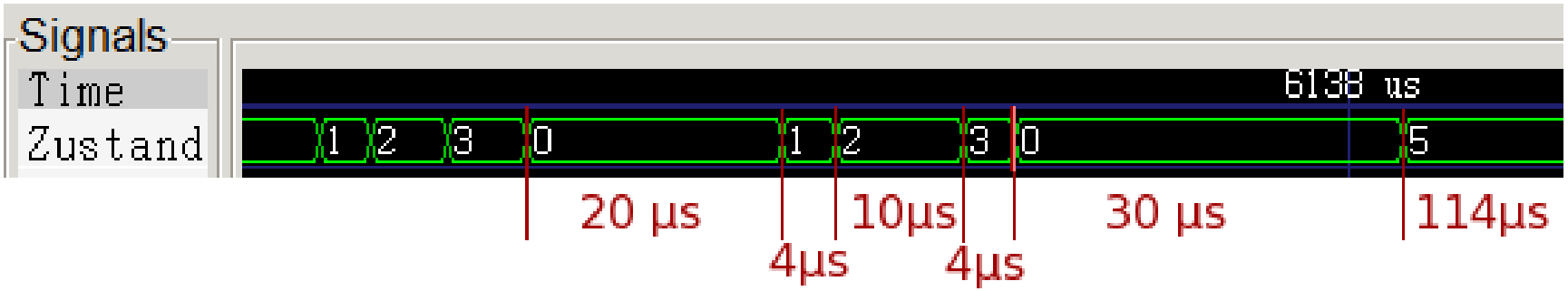
- HTerm starten. COM..., 9600 Baud, 8n1. Connect, »asdf« senden.

Erwarteter Signalverlauf:





- Aufzeichnung von 8,192 ms (2 ms×4048 Werte).
- Trigger (Zustand=5) nach $\frac{6}{8}$ der Aufzeichnungszeit. Warum?
- Eine Byte-Übertragung dauert etwa 1 ms und die Verarbeitung einer neuen Nachricht 0,5 ms.
- Abarbeitungszeiten der Schrittfunktionen:



Aufgaben

Aufgabe 5.1: Bluetooth-Treiber

Schreiben Sie in Anlehnung an den Treiber »comsf_pc« einer Treiber »comsf_bt« für die Bluetooth-Kommunikation. Header:

```
#ifndef COMFS_BT_H_
#define COMFS_BT_H_
#include <avr/io.h>
#define COM_BT_SMSG_LEN 4
#define COM_BT_RMSG_LEN 4
 void com_bt_init();              // Initialisierung
 void com_bt_step();              // Schrittfunktion
 uint8_t com_bt_get(uint8_t *msg); //Empfang
 uint8_t com_bt_send(uint8_t *msg);//Senden
#endif /* COMFS_BT_H_ */
```

indem Sie USART2 durch USART0 ersetzen. Test durch Ersatz der über USB getunnelten Verbindung durch die über Bluetooth.

Aufgabe 5.2: PC-Sonarabfrage

1. Schreiben Sie unter Verwendung der Treiber »comsf_pc« und »comsf_sonar« ein Programm, das in einer Endlosschleife
   - vom PC auf das Zeichen »s« wartet und
   - den zuletzt empfangenen Sonarwert als 2-Byte-Zahl zum PC sendet.

   Testen Sie das Programm mit dem HTerm.
2. Warum wird auf dem PC immer der Abstandswert der vorletzten Anfrage zurückgegeben?
3. Überlegen Sie sich einen Workaround, damit immer der aktuelle Wert an den PC gesendet wird.

Aufgabe 5.3: Chat-Programm

Schreiben Sie ein Programm, das vom PC über die Bluetooth-Verbindung empfangene Zeichen über die USB-Verbindung zurück sendet und umgekehrt.
Benutzen Sie das Programm zum wechselseitigen Übertragen von Zeichen

- vom HTerm auf einem Rechner zum HTerm auf einem anderen Rechner,
- zwischen gleichen Python-Programmen auf unterschiedlichen Rechnern.

Einer der Rechner verwendet dabei jeweils die Kabel- und der andere die Bluetooth-Verbindung.