Informatikwerkstatt, Foliensatz 4
IO-Hintergrundprozesse

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F4)

7. Dezember 2015

Inhalt Foliensatz 4

Wiederholung

Wiederholungsaufgabe 4.1

```
void com_pc_init(){
  // Initialisierung von USART2
  UCSR2C=0b110;                    // Übertragungsformat 8N1
  UBRR2=51;                        // 9600 Baud
  UCSR2B=(1<<RXEN2)|(1<<TXEN2);// Empf. und Sender ein
}
```

- Was bedeutet Übertragungsformat 8N1?
- Was ist die Baudrate?
- Wie lange dauert die Übertragung eines Bytes mindestens?

Wiederholungsaufgabe 4.2

1. Wozu dient und wie funktioniert nachfolgende Funktion?

```
#define NEW_DAT (UCSR2A & (1<<RXC2))
uint8_t getByte(){
  while (!NEW_DAT); //warte auf ein Byte
  return UDR2;          //Byte zurueckgeben
}
```

2. Vervollständigen Sie die nachfolgende Funktion zum Empfang einer vorzeichenfreien 2-Byte-Zahl.

```
uint16_t get2Byte(){
 uint16_t a;       //lokale Variable
 while (!NEW_DAT);//warte auf ein Byte




 return a;          //2-Byte-Wert zurueckgeben
}
```

Wiederholungsaufgabe 4.3

1. Wozu dient und wie funktioniert die nachfolgende Funktion?

```
void sendByte(uint8_t dat){
  while (!(UCSR2A & (1<<UDRE2)));// warte Puffer frei
  UDR2 = dat;                    // Byte uebernehmen
}
```

2. Wie ist die nachfolgende Funktion zu ändern, damit sie vorzeichenfreie 2-Byte-Zahlen versendet?

```
#define SENDBUF_FREE (UCSR2A & (1<<UDRE2))
void send2Byte(uint16_t dat){
 while (!SENDBUF_FREE);// warte bis Puffer frei




}return;
```

Wiederholungsaufgabe 4.4

Die Funktion soll über UART2 beginnend mit null bei jedem Aufruf den um eins erhöhten Bytewert senden (Funktion mit Gedächtnis). Bitte vervollständigen:

```
#define SENDBUF_FREE (UCSR2A & (1<<UDRE2))
#define SBUF UDR2

void sendZaehlwert(){




}
```

Wiederholungsaufgabe 4.5

- Wie funktioniert das nachfolgende Programm?

```
int main(){
  com_pc_init();
  uint8_t i, dat, *ptr, text[]="Ausgabetext\n";
  while(1){
    dat = getByte();
    for (i=0; i<(dat & 0b111), i++){
      sendByte(text[i]);
      ptr++;
    }
  }
}
```

- Welche Zeichenfolgen empfängt der PC, wenn er im Abstand von 1 Sekunde die Zeichen 'A', 'D', und 'F' sendet?

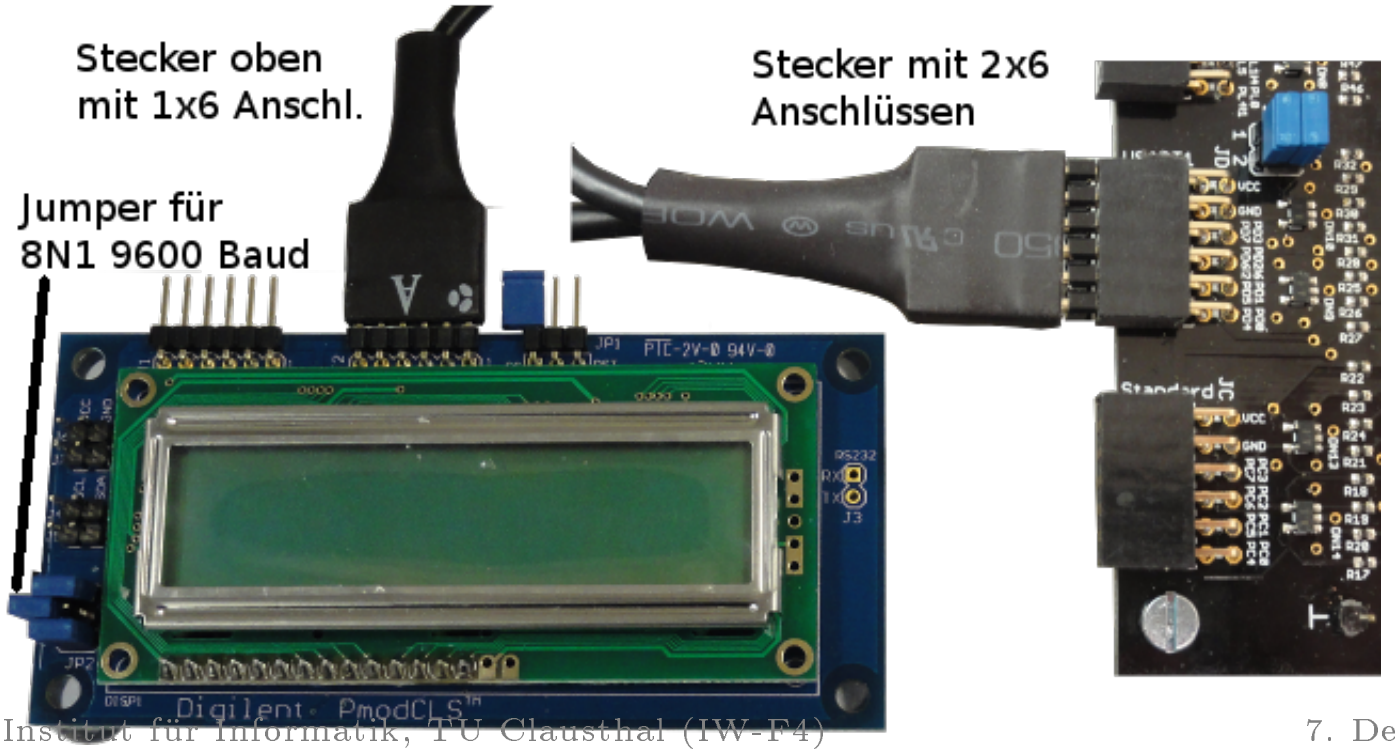
Weitere Geräte mit USART

LC-Display

LC-Display

Das LC-Display hat eine serielle Schnittstelle am Stecker J2, von der nur »Schreiben« benötigt wird:

- MR-Board JD oben (UART 1) über Kabel an LCD J2
- für Protokoll UART, 8N1, 9600 Baud auf LCD Jumper JP2 MD0 und MD2 stecken.



Testprogramm »test_lcd.c«

```
void lcd_init(){          //Initialisierung USART1
  UCSR1C = 0b110;         //Übertragungsformat 8N1
  UBRR1  = 51;            //9600 Baud
  UCSR1B = (1<<TXEN1); //Sender ein
}
uint8_t LCD_dat[]="\x1B[0h\x1B[j" //7-Zeichen-Init.[1]
                  "A=... Err:......" // 1. Zeile (16 Zeichen)
                  "S:... Inp:......";// 2. Zeile (16 Zeichen)
```

[1] "\x1B[0h" – Zeilenumbruch nach 16 Zeichen; "\x1B[j" – Anzeige löschen und Cursor auf erstes Zeichen; siehe auch PmodCLS_rm.pdf.

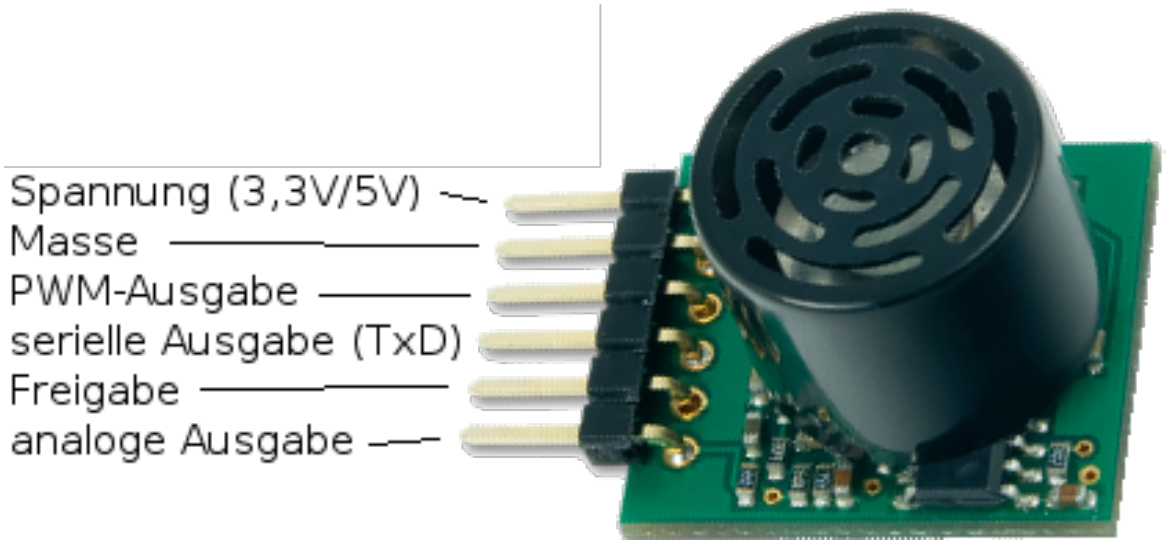
```
uint8_t idx;                              //Indexvariable
int main(void){
  lcd_init();                             //USART1 initialisieren
  for(idx=0;idx<39;idx++){                //für alle 39 Zeichen
    while (!(UCSR1A&(1<<UDRE1)));//warte, bis Puffer frei
    UDR1 = LCD_dat[idx];                  //Zeichen in Puffer
  }
}
```

Das LC-Display dient im Weiteren für die Anzeige von Fahrzeugsteuerzuständen, Fehlern, Sensorwerten, ...

- Projekt »F4-1_test_lcd\test_lcd« öffnen.
- Programm übersetzen und ausführen.
- LCD-Ausgabe kontrollieren.

Sonar-Sensor

Sensormodul PmodMAXSONAR



- PmodMAXSONAR über Y-Kabel an JD unten (UART1) anstecken. JDX auf »gekreuzt (=)«
- Wie auf der nächsten Folie gezeigt, Multimetereingang an Pin 1 und Multimetermasse (COM) an einen Masseanschluss (⊥) des Mikrorechner-Boards.
- Spannung zuschalten.

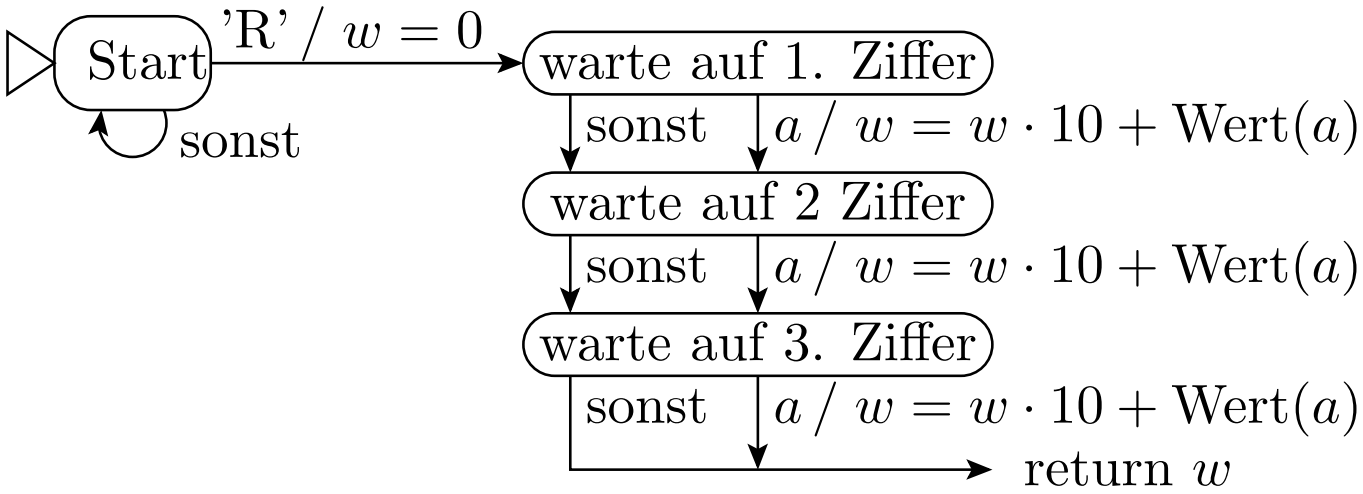
- Projekt »F4-2_test_sonar\test_sonar«, übersetzen, im Debugger starten und anhalten.
- Zur Aktivierung PD5 (Freigabesignal) im Debug-Modus als Ausgang und auf eins setzen.
- Hand im Anstand $> 15\,\mathrm{cm}$ vorm Sensor bewegen.

Spannung am Multimeter: $3{,}3\,\mathrm{V} \cdot x/512" \approx 2{,}5\,\frac{\mathrm{mV}}{\mathrm{cm}}$. Messbereich 6" bis 254" (1" = 2,54 cm). Mindestabstand $\approx 15\,\mathrm{cm}$. Auflösung 1" (" – Zoll, engl. Inch).

Der Mikrorechner empfängt an UART1 eine Zeichenfolge mit dem Format:

$$\text{'R'}\ z\ z\ z\ \text{'\textbackslash 13'}$$

('R' – ASCII-Zeichen; z – Ziffer 0..9 als ASCII-Zeichen; '\13' – Zeilenumbruch). Akzeptorautomat Decodierung Abstandswert:



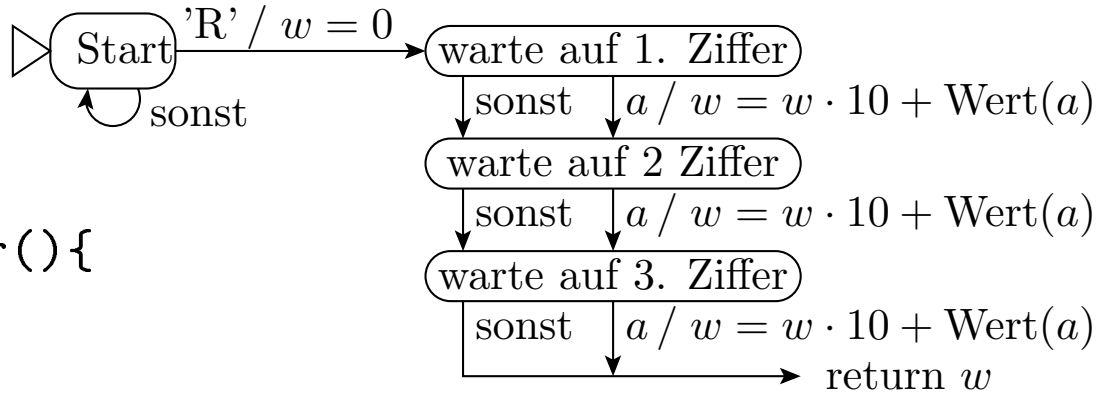
Testprogramm »test_sonar.c«

```
int main(){
  sonar_init();                 //Sensor initialisiern
  DDRJ = 0xFF;                  //LED-Ausgabe initialisiern
  while(1){
    uint16_t s = getSonar();//Wert abholen
    PORTJ = s & 0xFF;           //Wert ausgeben
  }
}
```

- Initialierung des seriellen Empfangs und des Sensors:

```
#define NEW_DAT (UCSR1A & (1<<RXC1))
void sonar_init(){              //Initialisierung von USART1
  UCSR1C=0b110;                 //Übertragungsformat 8N1
  UBRR1=51;                     //9600 Baud
  UCSR1B=(1<<RXEN1);            //Empfänger ein
  DDRD  |= 1<<PD5;              //PD5 Ausgang
  PORTD |= 1<<PD5;              //Sonar einschalten
}
```

- Empfangs-automat:




```
uint16_t getSonar(){
  uint8_t dat, i;
  while (1){
    while (!NEW_DAT);     //Warte auf Daten
    if (UDR1=='R') break;//Warte auf Zeichen 'R'
  }
  uint16_t w = 0;
  for (i=0;i<3; i++){     //Wiederhole 3x
    while (!NEW_DAT);     //Warte auf Daten
    dat = UDR1;
    if (dat>='0' && dat<='9')
      w = (w*10) + (dat-'0');
  }
  return w;
}
```

- Programm »test_sonar« starten und LED-Ausgabe kontrollieren.

IO-Hintergrundprozesse

IO-Hintergrundprozesse

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

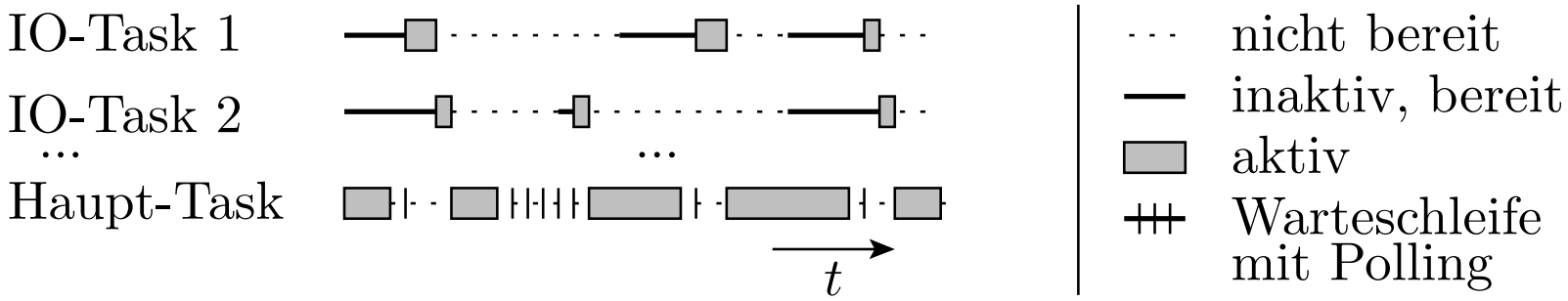
Geplante Software-Architektur mit nebenläufigen[3] IO-Tasks:

- Treiber für alle IO-Tasks, die nur bei Bereitschaft abgearbeitet werden (Tasks ohne Warteschleifen).
- Haupt-Task mit blockierungsfreien IO-Operationen und einer zentralen Warteschleife, in der zyklisch alle IO-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 mehr hat, fragt er reihum die IO-Tasks ab, ob sie bereit sind. Wenn ja, Abarbeitung des IO-Tasks bis zum Start der nächsten Ein- oder Ausgabe.
- Falls kein IO-Task bereit ist, wiederholt der Haupt-Task die Abfrage zyklisch.
- Nach Abarbeitung aller bereiten IO-Tasks hat der Haupt-Task möglicherweise wieder Daten für seine Fortsetzung.
- Voraussetzung kurze Task-Schritte.

Treiber für IO-Tasks

Ein IO-Task sei ein Objekt mit einer Schrittfunktion ohne Warteschleifen:

- wenn nicht bereit, Rücksprung,
- wenn bereit, Schritt abarbeiten und Rücksprung.

```
//globale private Variablen des Treibers inkl. Zustand
void Schrittfunktion_IO_Task_x(){
  if (!<bereit>[4]) return;
  switch (<Zustand>): //Fallunterscheidung nach Zustand
   case <Zustand 1>:  //für den ersten Zustand
     ...              //Anweisungsfolge incl. Zustand
     return;          //weiterschalten und Rücksprung
   case <Zustand 2>:  //für den zweiten Zustand
     ...
  }
}
```

[4] Z.B. für »UARTi Byte empfangen« SFR-Bit »RXCi==1«.

Ein IO-Task benötigt weiterhin

- 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;         // erfolgreicher Rückkehr
}
```

Hauptprogramm

```
int main(){
 //Variablenvereinbarungen für das Hauptprogramm
 ...
 init_Task_x(<Initialisierungswerte>);
 init_Task_y(<Initialisierungswerte>); ...
 while (1){
  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 Abbruch
    case ...:            //für den zweiten Zustand
  }
}
```

Exp. mit 3 Treibern

Treiber LCD-Monitor

LCD-Monitor und sein Treiber (comsf_lcd.c[5])

Das LCD soll zur Darstellung zeitveränderlicher Programmzustände, Sensorwerte, Fehlerzähler, ... genutzt werden.

- Die Schrittfunktion soll zeichenweise zyklisch den als private Daten gespeicherten Text an das LC-Display senden.

[5] Projekt F4-3_comsf\comsf.

Öffnen Sie die Datei »comsf_lcd.h«!

- Fehlerzähler erhöhen:

```
void lcd_incErr(uint8_t pos);
Erhöhung des Zeichens der LCD-Position »pos« in
der Reihenfolge 1, 2, .., a, b, ... bis Endwert z.
```

- Einzelzeichen auf LCD ausgeben:

```
void lcd_disp_chr(uint8_t c, uint8_t pos);
Ausgabe von »c« auf Position »pos«.
```

- Text auf LCD ausgeben:

```
void lcd_disp_str(uint8_t *str, uint8_t pos, uint8_t len);
Ausgabe des Textes der Länge »len« ab
Speicheradresse »str« ab LCD-Position »pos«.
```

- 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 und Schrittfunktion:

```
//globale private Daten
    uint8_t LCD_dat[32]; //Ausgabetext
    uint8_t lcd_idx;     //Indexvariable
```

- Die Schrittfunktion 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(){//Schrittfunktion
  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 des Treibers:

- USART1 8N1, 9600 Baud. Sender ein.
- Initialisierzeichenfolge »\x1B[0h\x1B[j« senden.
- Hintergrundtext initialisieren.

```
void lcd_init(uint8_t *text){
  //USART1 mit 8N1, 9600 Baud initialisieren
  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];
  }
  //Initialisierung des Hintergrundtextes
  for (lcd_idx=0; lcd_idx<32; lcd_idx++)//Übernahme des
    LCD_dat[lcd_idx] = text[lcd_idx];   //Init.-Texts
  lcd_idx = 0;                          //Index auf Feldanfang
}
```

Fehlerzähler

Ziel ist die Signalisierung von Fehlfunktionen während der Programmabarbeitung, z.B. Verlust empfangener Daten, Division durch Null, ... Dazu soll das Zeichen mit der Position »pos« in der Reihenfolgen ». 1 2 ... 9 a b ... z« hochzählen und im Zustand »z« (zu viele Fehler) 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;
}
```

Zeichen und Texte im Ausgabetext ändern

- Zeichen eignen sich gut für die Darstellung von Automatenzuständen.

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

- Eine Textausgabe ist eine len-fache Zeichenausgabe:

```
//Text auf LCD ausgeben
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--;
  }
}
```

Zahlenwert (über-) schreiben

```
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. Ziffer durch '?'
}
```

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

Treiber PC-Komm.

Öffnen Sie die Datei »comsf_pc.h«!

Einstellung Sende- und Empfangspuffergröße im Header:

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

Außer der Initialisierungs- und Schrittfunktion stellt der Treiber für das übergeordnete Programm Funktionen bereit zum

- Lesen einer Nachricht

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

- Versenden einer Nachricht

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

- Lesen des zuletzt empfangenen Bytes

  ```
  uint8_t com_pc_last_byte();
  ```

- und Löschen empfangener Bytes (Empfangspuffer leeren)

  ```
  void com_pc_rmsg_clr();
  ```

bereit.

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
```

Initialisierung

```
void com_pc_init(){
  //Initialisierung fur USART2 8N1, 9600 Baud
  UCSR2C=0b110;                  //8N1
  UBRR2=51;                      //9600 Baud
  UCSR2B=(1<<RXEN2)|(1<<TXEN2); //Sender und Empf. ein
  //Initialisierung von Empfangs- und Sendepuffer
  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
}
```

Funktionen 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 kopieren
  ridx = 0;                  //Empfangspuffer leeren
  return 1;                  //Rücksprung "Daten erhalten"
}
```

- Letztes empfangene Byte lesen (wird später für Sofort-Kommandos wie »Auto sofort anhalten« genutzt):

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

Nicht blockierendes versenden einer Nachricht:

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

Testrahmen und Ausprobieren erst gemeinsam mit dem dritten Treiber, der die Sonar-Abstandswerte bereitstellt.

Treiber Sonar-Sensor

Öffnen Sie die Datei »comsf_sonar.h«!

Initialisierungsfunktion:

```
void sonar_init();
```

Schrittfunktion:

```
void sonar_step();
```

Abholfunktion für den empfangenen Abstandswert:

```
uint8_t sonar_get(uint16_t *sptr);
Wenn der Treiben 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«.
```

Öffnen Sie die Datei »comsf_sonar.c«!

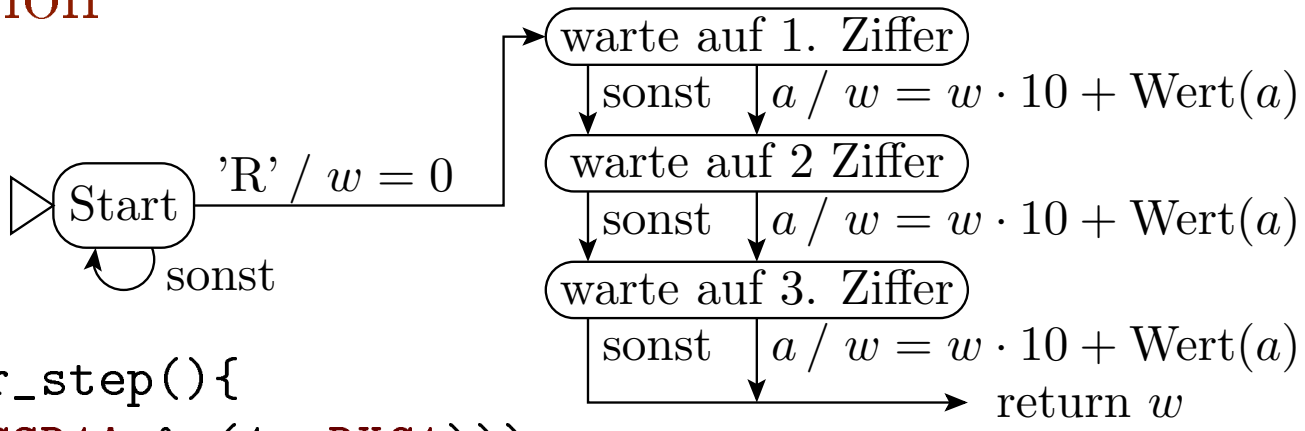
Private Daten sind der Abstandswert und der Treiberzustand:

```
uint16_t snr_val; uint8_t  snr_state;
```

Die Initialisierung schaltet auch den Sensor ein:

```
void sonar_init(){   // Initialisierung USART1-Empf.
  UCSR1C=0b110;      // Übertragungsformat 8N1
  UBRR1=51;          // 9600 Baud
  UCSR1B|=(1<<RXEN1);// Empfänger ein
  DDRD  |= 1<<PD5;   // PE1 Ausgang
  PORTD |= 1<<PD5;   // Sonar einschalten
  snr_state =0;
}
```

Schrittfunktion




```
void sonar_step(){
  if (!(UCSR1A & (1<<RXC1)))
    return;              //kein neues Zeichen, beenden
  int8_t dat = UDR1; //sonst Zeichen lesen
  if (dat=='R'){       //wenn 'R', Zustand
    snr_state = 1;     //"warte 1. Hexziffer'
    snr_val   = 0;     //Wert mit 0 initia-
    return;            //lisieren
  }  //wenn »warte auf Ziffer« Wert * 10 + Ziffernwerk
  if (snr_state>0 && snr_state<4){
    if (dat>='0' && dat<='9')
      snr_val = (snr_val*10) + (dat-'0');
  } snr_state++;
}
```

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>=4) {// wenn neuer Wert da
    *sptr = snr_val; // ausgeben
    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

Einstellung Sende- und Empfangspuffergröße in »comsf_pc.h«:

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

An die Applikation anzupassen.

Initialisierungsfunktionen:

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

Aufruf im Initialisierungsteil des Programms.

Schrittfunktionen:

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

Zyklischer Aufruf in der Hauptschleife.

Anwenderfunktionen des LCD-Treibers:

- Fehlerzähler erhöhen:

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

- Einzelzeichen auf LCD ausgeben:

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

- Text auf LCD ausgeben:

```
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);
```

Anwenderfunktionen des Treibers für die PC-Kommunikation:

- Lesen einer Nachricht vom PC:

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

- Versenden einer Nachricht an den PC:

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

- ...

- Lesen des zuletzt empfangenen Bytes:

```
uint8_t com_pc_last_byte();
```

- Löschen empfangener Bytes (Empfangspuffer leeren):

```
void com_pc_rmsg_clr();
```

Anwenderfunktionen des Treibers für den Sonarsensor:

- Abstandswert lesen:

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

Aus Sicht der objektorientierten Programmierung sind Treiber Datenobjekte mit Bearbeitungsmethoden und exklusiver Hardware-Nutzung.

Testrahmen »test_comsf.c« für die drei Treiber

Anzeige auf dem LCD:

- Sonarwert,
- 4-Byte-Empfangsnachricht,
- letztes empfangenes Zeichen,
- Anzahl der Abstandsmessungen durch 50,
- Anzahl der empfangenen Nachrichten,
- 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++;   //Zähler Sonarmessungen 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:

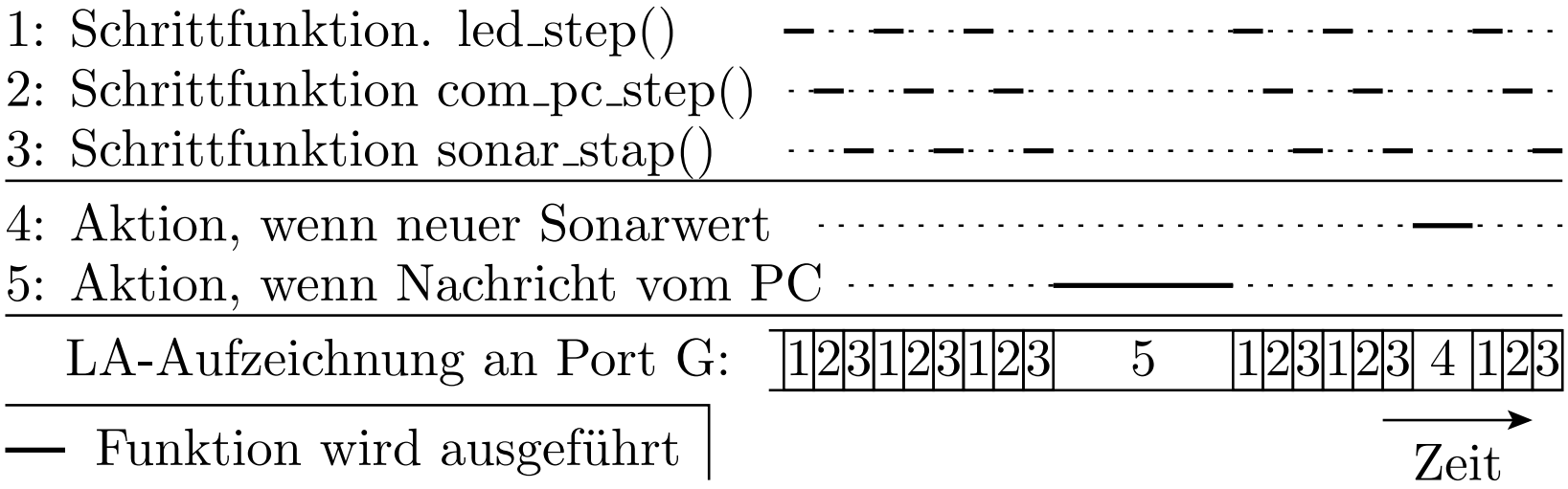
```
lcd_disp_chr(com_pc_last_byte(), LCP_LBYTE);
```

Ausprobieren des Testbeispiels (test_comsf)

Projekt »F4-3_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.

Kontrolle der Zeitabläufe mit dem Logikanalysator

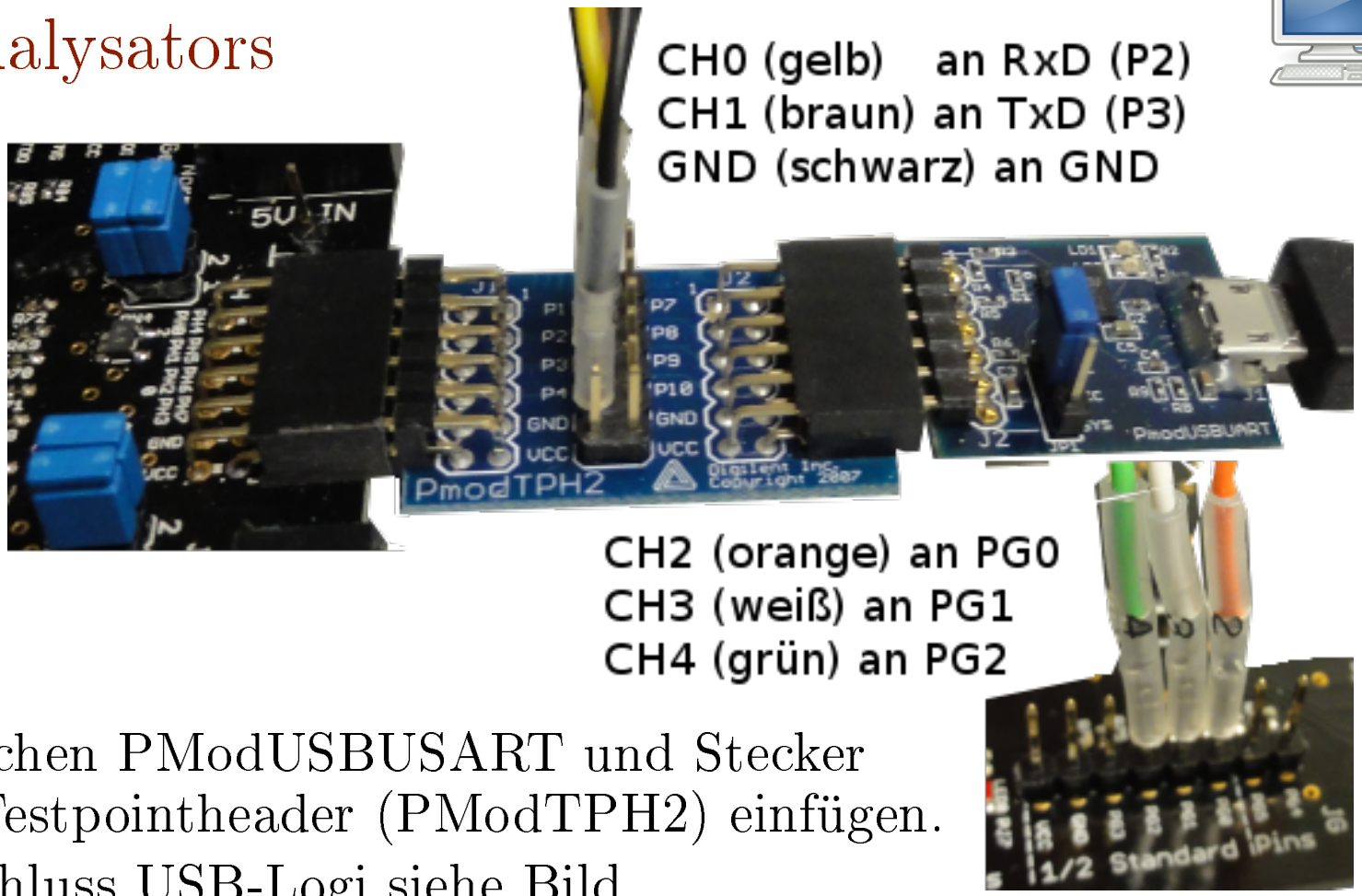


Zur Visualisierung der Dauer und Reihenfolge der Teilaufgaben:

- 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 +1
    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>
```

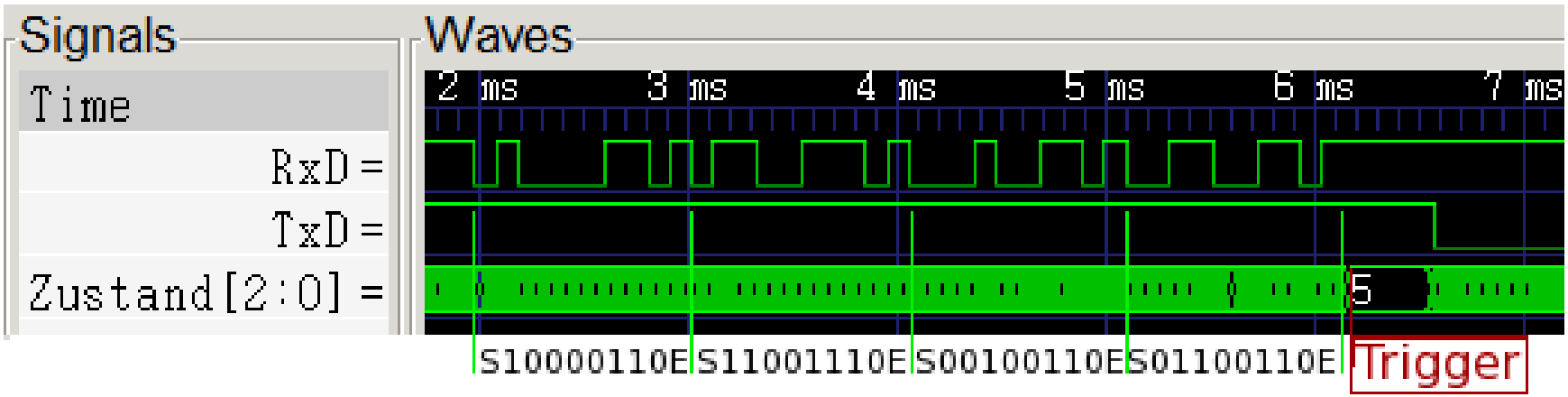
Experiment

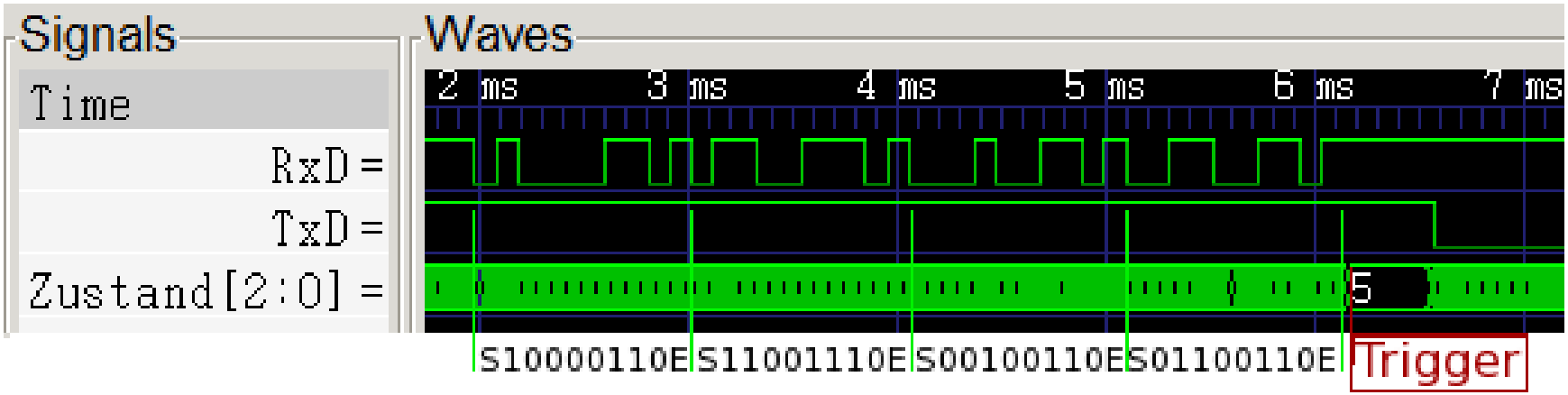
- Programm »F4-3_comsf\test_cmsf« läuft.
- Windows-Konsole »cmd« starten. In das Verzeichnis ...\USBLOGI wechseln.
- Logikanalysator starten mit:

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

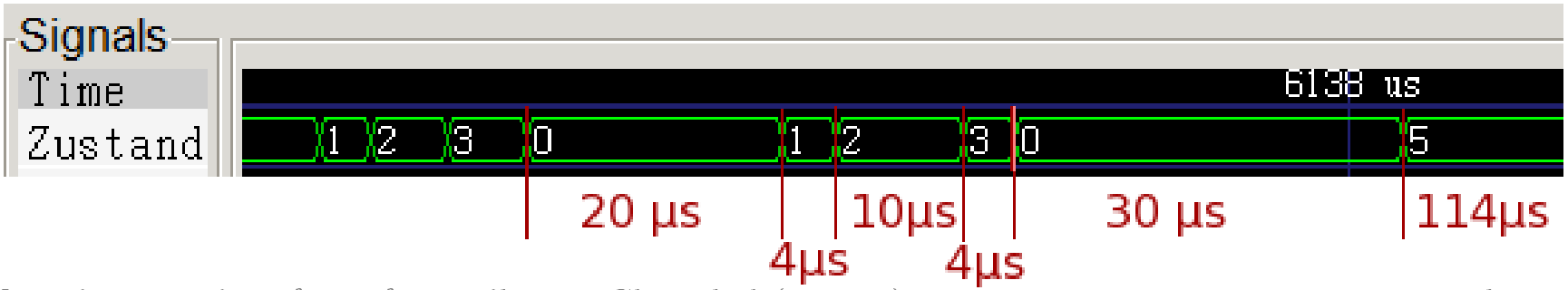
- HTerm starten. COM..., 9600 Baud, 8n1. Connect, »asdf« senden.

Erwarteter Signalverlauf:





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



Aufgabe 4.1:

Ändern Sie das Programm »test_lcd.c« (Folie 12) so ab, dass folgender Text angezeigt wird:

```
Zeile 1: "abcdefghijklmnop"
Zeile 2: "ABCDEFGHIJKLMNOP"
```

Aufgabe 4.2:

Fassen Sie die Testprogramme für den Sonarsensor und das LC-Display so zusammen, dass die Zeichenfolge vom Sensor

```
"R", <Ziffer>, <Ziffer>, <Ziffer>
```

direkt ohne Zwischenumwandlung in eine Zahl auf eine bestimmte Position des LC-Displays geschrieben wird.

Aufgabe 4.3: Compiler-Optimierung und Laufzeit

Wiederholen Sie die Laufzeituntersuchungen ab Folie 60 mit einer höheren Compiler-Optimierung. Wie ändern sich:

- die Abarbeitungszeit von Phase 5,
- die Byte-Übertragungszeiten und
- die Abarbeitungszeiten für die Schrittfunktionen?

Aufgabe 4.4: LCD-Treiber mit LA testen

Schrittfunktion des LCD-Treibers um folgende Debug-Ausgaben erweiteren:

```
void lcd_step(){
  if (UCSR1A&(1<<UDRE1)){      //wenn Puffer frei
    PORTA = LCD_dat[lcd_idx]; //Debug-Ausgabe
    PORTK = lcd_idx;          //Debug-Ausgabe
    UDR1 = LCD_dat[lcd_idx];  //schicke nächstes
    lcd_idx++;                //Zeichen
     if (lcd_idx>=32) lcd_idx = 0;
  }
}
```

- Mit dem USB-Logi kompletten Zyklus aller 32 Byteausgaben aufzeichnen. Aufzeichnungsrate 50.000 Werte/s.
- USB-Logi an Bits von Port A und K anstecken (mit Doppelstiftleisten).

- Port A und K in der Init-Funktion als Ausgänge initialisieren.
- xml-File anpassen:

```
<samplerate>50000</samplerate>    #50.000 Werte/s
  <pretrigger>1</pretrigger>       #Hälfte Vortrigger-
  <signals>                        #aufzeichnung
   <signal name="RxD"> <ch>0</ch> </signal> #gelb
   <signal name="TxD"> <ch>1</ch> </signal> #braun
   <signal name="lcd_idx">         #Port K
     <ch>2</ch><ch>3</ch> ... <ch>9</ch>
   </signal>
   <signal name="lcd_dat">         #Port K
     <ch>10</ch><ch>11</ch> ... <ch>17</ch>
   </signal>
  <trigger when="A">               #Trigger bei
    <A> <ch when="high">2</ch>     #lcd_idx==15
        ... </A>
  </trigger>
```

- Programm neu übersetzen und starten.
- Logikanalysator mit angepasster xml-Datei starten.

Probieren Sie das Programm weiterhin aus

- mit geänderter Textausgabe auf das LCD,
- mit geänderten Puffergrößen,
- mit geänderten Ausgabepositionen, ...