



Informatikwerkstatt, Foliensatz 7

Treiber mit Schrittfunktionen

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F7)

28. November 2022



Inhalt:

Wiederholung

EA-Hintergrundprozesse

Treiber LCD-Monitor

Treiber PC-Kommunikation

Treiber Sonar-Sensor

Test der Treiber

Ablaufkontrolle mit LA

Aufgaben

Interaktive Übungen:

- 1 Test der Treiber mit Schrittfunktion (F7-comsf/comsf)
- 2 Test der Zeitabläufe mit dem Logikanalysator (USBLOGI)



Wiederholung



W7.1: Kommunikation mit dem PC



Entwickeln Sie ein Programm mit, PModUSBUSART an JH, dem Schaltermodul an JA und dem 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_ */
```

Es soll

- in einer Endlosschleife auf ein Bytes vom PC warten,
- dessen Wert auf die LEDs ausgeben und
- den Wert plus Schalterwert modulo-128¹ zurücksenden.

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



1. Wiederholung

```
#include "com_pc.h"
uint8_t b, c, d; // Variablen
int main(){
    . . . . . // Port A: Schaltereingabe
    . . . . . // Port J: LED-Ausgabe
    . . . . . // USART2: 8n1, 9600 baud
    while(..){ // Endlosschleife
        b = . . . . . // b = empfangenes Byte
        . . . . . // Ausgabe b an die LEDs
        c = . . . . . ; // c = Bytewert + Schalterwert
        d = . . . . . ; // d = mod128(c) (Bit 7 löschen)
        . . . . . ; // Ergebnis zurücksenden
    }
}
```



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
    }
}
```



W7.2: 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.



W7.3: 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 Signale, 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 Task² auf Abschluss einer EA-Operation wartet, einen anderen Task, der bereit ist, weiter abarbeiten.

Geplante Software-Architektur mit nebenläufigen³ EA-Tasks:

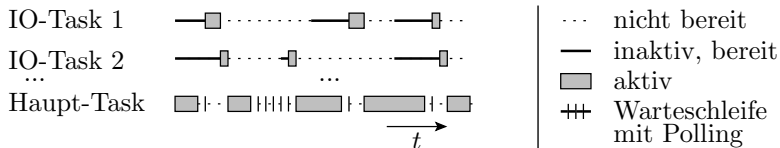
- Treiber für alle EA-Tasks, die nur bei Bereitschaft abgearbeitet werden (Tasks ohne Warteschleifen).
- Haupt-Task mit blockierungsfreien EA-Operationen und einer zentralen Warteschleife, in der zyklisch alle EA-Tasks auf Bereitschaft abgefragt werden.

²Task: Abzuarbeitende Aufgabe.

³Nebenläufig: 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 Treiber ist ein Programmobjekt mit

- privaten Daten,
- einer Initialisierungsfunktion für private Daten und zugeordnete Hardware zum Aufruf beim Programmstart,
- Funktionen zum blockierungsfreien Lesen und Schreiben von Ein- oder Ausgabedaten für den 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;      //weitschalten und Rückspr.
        case <Zustand 2>: //für den zweiten Zustand
            ...
    }
}
```



2. EA-Hintergrundprozesse

- 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

Treiber LCD-Monitor (`comsf_lcd.h/c4`)



- 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 den anderen Programmteilen vom Treiber zu Verfügung gestellten »öffentlichen Funktionen« schreiben blockierungsfrei in den privaten Textspeicher.

⁴Testbeispiel gemeinsam mit dem Treiber für die PC-Kommunikation und den Ultraschallempfänger siehe Projekt P07\F7-comsf\comsf.



Öffentlichen Funktionen (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);
```



Implementierung in »comsf_lcd.c«

Private Daten:

```
uint8_t LCD_dat[32];           //Ausgabertext
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 der Indexwert.

```
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
```



3. Treiber LCD-Monitor

■ 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
}
```

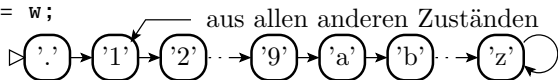


3. Treiber LCD-Monitor

- Fehlerzähler erhöhen

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;
}
```





3. Treiber LCD-Monitor

- 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--;  
    }  
}
```




3. Treiber LCD-Monitor

- Ausgabe von Zahlenwerten (z.B. Sensorwerten)

```
void lcd_disp_val(uint32_t val, uint8_t pos,
                 uint8_t len){
while (len){
    len--;
    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 Testsbeispiel für den Treiber folgt ab Folie 42.



Treiber PC-Kommunikation

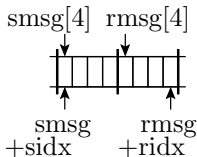


Treiber PC-Kommunikation (comsf_pc.h/c)

- 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();           //Schrittfunktion
uint8_t com_pc_get(uint8_t *msg); //Empfangsfunktion
uint8_t com_pc_send(uint8_t *msg); //Sendefunktion
uint8_t com_pc_last_byte(); //letztes empf. Bytes
```



- wenn leer, 4 byte in `smsg[4]` schreibbar
- dann byteweise mit Schrittfkt. versenden

- wenn voll, 4 byte aus `rmsg[4]` lesbar
- dann byteweise mit Schrittfkt. füllen



4. Treiber PC-Kommunikation

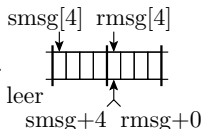
Öffnen Sie die Datei »comsf_pc.c«!

■ 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:

- Sendepuffer leer
- Empfangspuffer leer



■ Initialisierungsfunktion (USART2, PmodUSBUSART an JH) :

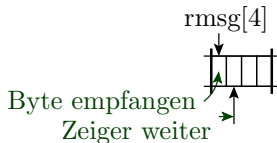
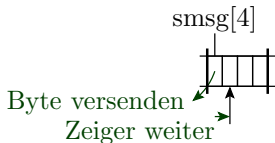
```
void com_pc_init(){
    UCSR2C=0b110;           //8N1
    UBRR2=51;               //9600 Baud
    UCSR2B=(1<<RXEN2)|(1<<TXEN2); //Send + Empf. ein
    //Initialisierung der Puffer
    ridx = 0;               //Empfangspuffer leer
    sidx = COM_PC_SMSG_LEN; //Sendepuffer leer
}
```



4. Treiber PC-Kommunikation

■ 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 Sendedaten bereit, diese versenden  
    }  
}
```





4. Treiber PC-Kommunikation

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





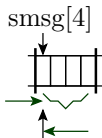
4. Treiber PC-Kommunikation

■ Sendefunktion (nicht blockierend)

Wenn Platz, Nachricht von »COM_PC_RMSG_LEN=4« Bytes in den Sendepuffer kopiert und Rücksprung mit 1 (OK).

```
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 Nachrichtenanfang
    sidx = 0; //Sendepuffer voll
    return 1; //Rücksprung mit
} // "Daten übergeben"
```

alle 4 zu versendende Werte eintragen
Zeiger für den Empfang auf Anfang



Ein Testsbeispiel für den Treiber folgt ab Folie 42.



Treiber Sonar-Sensor



Nur zwei öffentliche Funktionen (comsf_sonar.h)

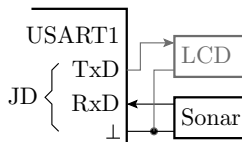
Der Treiber erwartet den Sonarsensor gemeinsam mit dem LC Display am Secker JD und stellt bereit:

- Initialisierungsfunktion:

```
void sonar_init();
```

- 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);
```





Treiberimplementierung (comsf_sonar.c)



Private Daten: Abstandswert und der Treiberzustand.

```
uint16_t snr_val; // Abstand w in Zoll
uint8_t snr_state; // Automatenzustand Z<i>
```

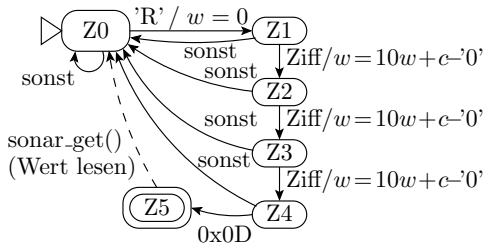
Die Initialisierung schaltet auch den Sensor ein:

```
void sonar_init(){ //Initialisierung USART1
    UCSR1C=0b110; //Übertragungsformat 8N1
    UBRR1=51; //9600 Baud
    UCSR1B|=(1<<RXEN1); //Empfänger ein5
    DDRD |= 1<<PD5; //PD5 Ausgang
    PORTD |= 1<<PD5; //Sonar einschalten
    snr_state =0;
}
```

⁵Sender nicht aus.



Schrittfunktion

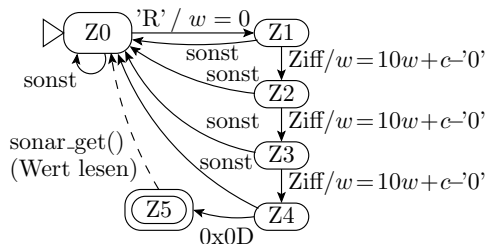


```

void sonar_step(){
    if (!(UCSR1A & (1<<RXC1))) //wenn kein neues
        return;                //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
}
  
```



5. Treiber Sonar-Sensor



// "Ziff"-Kanten

```

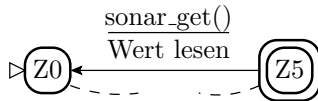
else if (snr_state > 0 && snr_state < 4 && dat >= '0'
        && dat <= '9'){
    snr_val = (snr_val * 10) + (dat - '0');
    snr_state++;
    return;
}
else if // Kante von Z4 nach Z5
        (snr_state == 4 && dat == 0x0D){snr_state = 5;}
else if (snr_state < 5)
    snr_state = 0; // "sonst"-Kanten
}
  
```



Abholfunktion für den empfangenen Abstandswert

Bei Abholen des Wertes wird der Zustand zurückgesetzt, 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



Alle zu testenden Funktionen

- Initialisierungsfunktionen:

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

- Schrittfunktionen:

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

- Die Lesefunktion für den Sonarwert:

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




6. Test der Treiber

- 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);
```

Puffergrößen im Testbeispiel: 4 Byte

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



Funktionstest⁶ »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 empfangene Nachrichten.

Rücksenden nach jedem Nachrichteneingang:

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

⁶Programm, dass alle zu testenden Funktionen nutzt.



6. Test der Treiber

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 srval, 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.



6. Test der Treiber

- 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);
    msmg[0] = snrval >> 8; //Sensorwert und Messwert-
    msmg[1] = snrval & 0xFF; //nummer byteweise in den
    msmg[2] = sct >> 8; //string "msg" schreiben
    msmg[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 »F7-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 Fehlerzähler hinter »E:« erhöhen.
- Der Fehlerzähler muss in der Reihenfolge ». 1 2 ... 9 a b ... z« zählen und im Zustand »z« verbleiben.



Anmerkung:

Es ist schwierig, in einem größeren Projekt den Überblick über alle Funktionsabläufe zu behalten.

Vieles, was Sie im Studium lernen werden, zielt hauptsächlich darauf, für größere Projekte Entwurf, Test, Wartung, ... zu beherrschen:

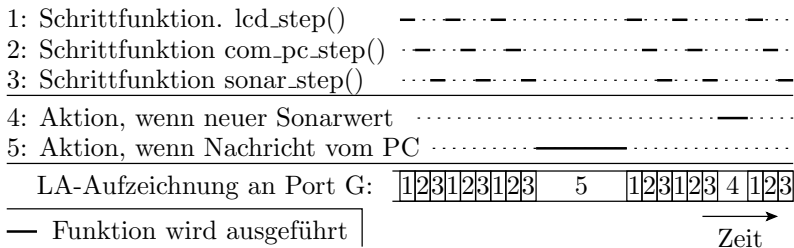
- Objektorientiertes Programmieren, Software-Architekturen,
- Entwicklungsabläufe, prüf- und wartungsgerechter Entwurf,
- Schichtenmodelle, Datenbanken, Betriebssysteme,
- Rechnernetze, verteilte Systeme, ...



Ablaufkontrolle mit LA



Kontrolle der Zeitabläufe mit dem Logikanalysator

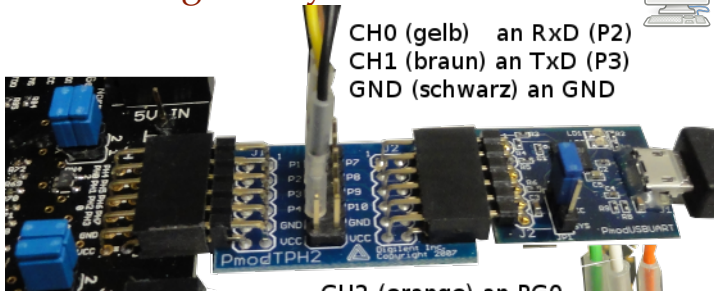


Zur Visualisierung der Dauer und Reihenfolge der Teilaufgaben:

- 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



CH0 (gelb) an RxD (P2)
CH1 (braun) an TxD (P3)
GND (schwarz) an GND

CH2 (orange) an PG0
CH3 (weiß) an PG1
CH4 (grün) an PG2



- Zwischen PModUSBUSART und Stecker JH Testpinthead (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>
```

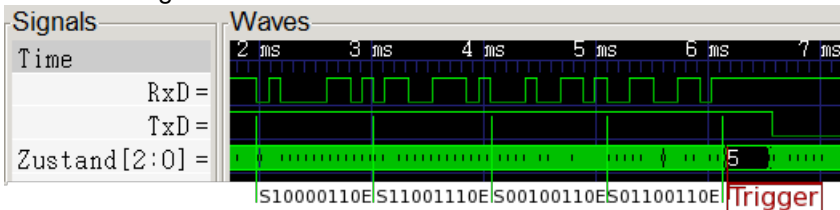


Experiment



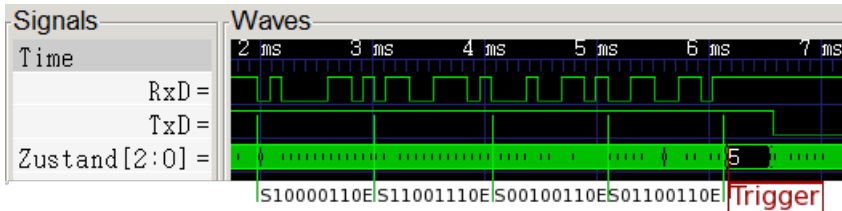
- Programm »...\\P07\F7-comsf\test_comsf« läuft.
- Windows-Konsole »cmd« starten. In das Verzeichnis\\P05\USBLOGI wechseln.
- Logikanalysator starten mit:
usb-logi test_comsf.xml test_comsf.sav
- HTerm starten. COM..., 9600 Baud, 8n1. Connect, »asdf« senden.

Erwarteter Signalverlauf:

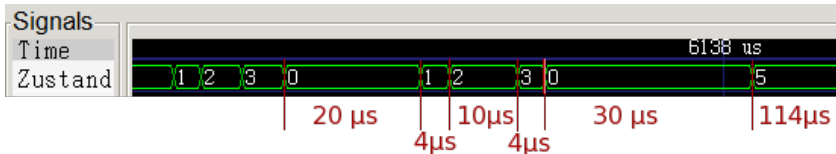




7. Ablaufkontrolle mit LA



- Aufzeichnung von 8,192 ms ($2 \text{ ms} \times 4048 \text{ Werte}$).
- 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:





Hausaufgabe

- Handout zum aktuellen Foliensatz noch mal lesen.
- Wiederholungsfragen auf dem nächsten Handouts beantworten.



Aufgaben



Aufgabe 7.1: 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 7.2: 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 7.3: 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.