



Informatikwerkstatt, Foliensatz 8

Interrupt-basierte Treiber

G. Kemnitz

Institut für Informatik, TU Clausthal (IW8)
1. Dezember 2016



Inhalt:

Wiederholung

Timer-Treiber (`comir_tmr`)

LCD-Treiber (`comir_lcd`)

Treiber PC-Kommunikation (`comir_pc`)

Treiber Ultraschallsensor (`comir_sonar`)

Testbeispiel mit allen Treibern

Aufgaben

Interaktive Übungen:

- 1 Treiber `comir_tmr` mit Testprogramm (`test_comir_tmr`)
- 2 Treiber `comir_lcd` mit Testprogramm (`test_comir_lcd`)
- 3 Treiber `comir_pc` und `comir_sonar` mit Testprogramm für alle Treiber (`comir`)



Wiederholung



Wiederholungsaufgabe 8.1: Interrupts



- Warum ist in einer ISR auszuschließen, dass sie sich selbst unterbricht?
- Warum dürfen ISR und unterbrochene Programmsequenz nicht dieselben Daten bearbeiten? Wie wird das für Programmteile sichergestellt, die von der ISR verarbeitete Daten lesen oder schreiben?



Lösung

- 1 Wenn sich eine ISR selbst unterbrechen kann und das auslösende Ereignisflag bleibt bei ISR-Aufruf gesetzt, unterbricht sich die ISR immer wieder selbst (Programmabsturz¹).
- 2 Wenn eine ISR Daten verändert, die die unterbrochene Programmsequenz gerade bearbeitet, kommt es oft zu inkonsistenten Treiberzuständen.

¹Bei jedem ISR-Aufruf werden Daten auf den Stack gelegt. Wenn der Stack voll ist, werden andere Daten überschrieben.



Wiederholungsaufgabe 8.2: Treiber



- 1 Wozu dienen die Initialisierungsfunktionen?
- 2 Wozu dienen die Schrittfunktionen und an welcher Stelle im Verarbeitungsfluss sind sie einzubinden?
- 3 Welche Aufgaben haben die Send- und die Get-Funktionen?
- 4 Was bedeutet blockierungsfreies Lesen oder Schreiben.
- 5 Warum müssen in einem Mehr-Task-System alle EA-Lese- und Schreibfunktionen blockierungsfrei arbeiten?



Lösung

- 1 Initialisierungsfunktionen konfigurieren die zugeordnete Hardware, legen private Daten an und initialisieren diese.
- 2 Schrittfunktionen testen auf EA-Ereignisse und führen für eingetretene EA-Ereignisse die dann erforderlichen Funktionen aus. Aufruf bisher am Anfang der Endlosschleife.
- 3 Die Send- und Get-Funktionen dienen in den Programmteilen, die die Treiber nutzen, zur Datenübergabe und Übernahme.
- 4 Blockierungsfrei bedeutet hier, dass die Funktion, wenn der Treiber noch nicht zur Datenübernahme bzw. Übergabe bereit ist, nicht wartet, sondern mit dem Status »Datenaustausch noch nicht möglich« zurückkehrt.
- 5 Blockierungsfrei, damit der Haupttask und mehrere EA-Tasks nebenläufig arbeiten können. Jeder Task wird abgearbeitet, sobald er bereit und der Rechner frei ist.



Timer-Treiber (comir_tmr)



Der Treiber »comir_tmr«

Das übergeordnete Programm der zu entwickelnden Fahrzeugsteuerung wird zeit- und ereignisgesteuerte Abläufe beschreiben, der Form:

- Fahre maximal 10 s geradeaus,
- wenn der Sensorwert größer 25, dann breche Fahrt ab, ...

Der Treiber »comir_tmr« nutzt Timer 1 und stellt über den Header comir_tmr.h folgende Funktionen bereit:

```
void tmr_init();           //Treiber initialisieren
uint32_t tmr_get();       //Zeitzähler lesen
```

Für vier unabhängige Timer-Kanäle:

```
void tmr_start(uint16_t tw, uint8_t nr); //Start
uint16_t tmr_restzeit(uint8_t nr); //lese Restzeit
```

($nr \in \{0, 1, 2, 3\}$ – Timer-Kanal; Zeitwert in 0,1 s-Schritten).



Private Daten und Initialisierungsfunktion

Private Daten:

```
uint32_t tmr_ct;           //Zähler Programmzeit
uint16_t tmr_array[4];    //4 Wartezeitähler
```

Initialisierung: Timer 1, CTC-Modus, Zähltakt $\frac{1}{32}$ MHz, OCR1A als Vergleichsregister, Ereignisperiode: $\frac{32 \cdot 3125}{1 \text{ MHz}} = 0,1 \text{ s}$:

```
#define WGM_CTC    0b0100    //Clear Timer on Compare
#define CS256      0b100     //Vorteiler 256
void tmr_init(){
    TCCR1A = WGM_CTC & 0b11; //Betriebsart & Zähltakt
    TCCR1B = (WGM_CTC & 0b1100) << 1 | (CS256 & 0b111);
    OCR1A  = 3125;           //Vergleichswert für 0,1s
    tmr_ct = 0;             //Laufzeitähler löschen
    TIMSK1 |= 1 << OCIE1A;  //Vergleichs-Int. A ein
}
```



Interruptroutine

Die ISR inkrementiert alle 100 ms den Programmzeitzähler und dekrementiert die Wartezeitzähler der Kanäle, die nicht null sind:

```
ISR(TIMER1_COMPA_vect){ //Vergleichs-Interrupt
    uint8_t idx;         //alle 100 ms
    tmr_ct++;           //Programmzeit zählen
    for (idx=0; idx<4; idx++) //für alle Wartezeitz.
        if (tmr_array[idx]) //wenn ungleich null
            tmr_array[idx]--; //abwärts zählen
}
```



Zugriffsmethoden für das Anwenderprogramm

```
uint32_t tmr_get(){           //Zeitzähler lesen
    return tmr_ct;
}
```

Für vier unabhängige Timer-Kanäle:

```
void tmr_start(uint16_t tw, uint8_t nr){//Start
    tmr_array[nr & 0b11] = tw;//Wartezeit schreiben
}
```

```
uint16_t tmr_restzeit(uint8_t nr){//Lesen der
    return tmr_array[nr & 0b11];    //Restzeit
}
```

Was ist an den Zugriffsmethoden falsch programmiert?

Werden möglicherweise Daten bearbeitet, die die ISR verwendet?



2. Timer-Treiber (comir_tmr)

- Zugriff mit unterbrechungsfreien Sequenzen:

```
void tmr_start(uint16_t tw, uint8_t nr){
    uint8_t tmp = TIMSK1;    //Int.-Zustand sichern
    TIMSK1 &= ~(1<<0CIE1A); //Vergleichs-Interrupt A aus
    tmr_array[nr & 0b11] = tw; //Wartezeit schreiben
    TIMSK1 = tmp;           //Int.-Zustand wiederherst.
}



uint32_t tmr_restzeit(uint8_t nr){ //Lese Restzeit
    uint8_t tmp = TIMSK1;    //Int.-Zustand sichern
    TIMSK1 &= ~(1<<0CIE1A); //Vergleichs-Interrupt A aus
    uint16_t z = tmr_array[nr & 0b11]; //Restzeit lesen
    TIMSK1 = tmp;           //Int.-Zustand wiederherst.
    return z;
}

uint16_t tmr_get(){ //Zeitzähler lesen
    uint8_t tmp = TIMSK1;    //Int.-Zustand sichern
    TIMSK1 &= ~(1<<0CIE1A); //Vergleichs-Interrupt A aus
    uint32_t z = tmr_ct;     //Uhrzeit zurückgeben
    TIMSK1 = tmp;           //Int.-Zustand wiederherst.
    return z;
}
```



Testbeispiel für der Treiber



- Timer-Kanal 0 soll die LED an PJ7 aller 0,7 s und Timer-Kanal 1 soll die LED an PJ6 aller 1,2 s invertieren.
 - Auf den LEDs an PJ0 bis PJ4 soll der Zeitwert in Sekunden ausgegeben werden.
-
- Projekt »F5-3_test_comir_tmr\test_comir_tmr« öffnen.
 - Übersetzen. Start im Debugger . Continue .
 - LED-Ausgaben kontrollieren.

Das Testprogramm:

```
int main(){
    tmr_init();           //Timer-Treiber initialisieren
    DDRJ = 0xFF;         //Port J LED-Ausgabe
    sei();                //Interrupts global ein
    while(1){            //Beginn Endlosschleife
```



2. Timer-Treiber (comir_tmr)

- Ablauf in der Endlosschleife:



```
if (!tmr_restzeit(0)){//wenn Kanal 0 abgelaufen
    PORTJ ^=0x40;          //LD6 invertieren
    tmr_start(12, 0);     //Kanal 0 mit 1,2 s init.
}
if (!tmr_restzeit(1)){//wenn Kanal 1 abgelaufen
    PORTJ ^=0x80;          //LD7 invertieren
    tmr_start(7, 1);     //Kanal 1 mit 0,7 s init.
}

//Zeit in s auf LED[4:0] ausgeben
uint8_t tmp = (tmr_get()/10) & 0x1F;
PORTJ = (PORTJ & ~0x1F) | tmp;
} //Ende der Endlosschleife
```

Anregungen zum Experimentieren:

- Die anderen Timer-Kanäle mitnutzen.
- Komplexere Blinksequenzen erzeugen.
- Schalter und LED-Module mit einbeziehen, ...



LCD-Treiber (comir_lcd)



Der Treiber »comir_lcd«

Umstellung des Treibers »comsf_lcd« auf Interrupts:

- Nutzt Sendeeinheit von USART1 (JD).
- Bei Initialisierung Sendepuffer-frei-Interrupt-Freigabe.
- ISR statt Schrittfunktion.
- BADISR mit dem letzten Anzeigezeichen als Fehlerzähler.

Unveränderte Übernahmen der Ausgabefunktionen aus dem Treiber »comsf_lcd«:

```
//Fehlerzähler erhöhen
void lcd_incErr(uint8_t pos);
//Einzelzeichenausgabe
void lcd_disp_chr(uint8_t c, uint8_t pos);
//Ausgabe eines Textes der Länge len
void lcd_disp_str(uint8_t *str, uint8_t pos,
                  uint8_t len);
```



3. LCD-Treiber (comir_lcd)

```
//Ausgabe eines Zahlenwertes  
void lcd_disp_val(uint32_t val, uint8_t pos,  
                  uint8_t len);
```

(pos – Schreibposition; len – Anz. der zu schreibenden Zeichen).

Unterbrechungssperren nicht erforderlich. Wenn das Schreiben auf den Puffer unterbrochen wird, erscheint im ungünstigsten Fall für wenige ms ein halb geänderter Text auf dem LCD.

Programmbeschreibung der Funktionen siehe Foliensatz IW5.



Private Daten und Initialisierung

Die privaten Daten sind dieselben wie beim Treiber comsf_lcd:

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

Die Initialisierungsfunktion aktiviert nur zusätzlich den Sendepuffer-frei-Interrupt:

```
void lcd_init(uint8_t *text){//LCD-Treiber init.
    ... //genau wie in comsf USART1 initialisieren
    ... //LC-Display initialisieren
    ... //Initialisierung des Hintergrundtextes
        //zusätzlich Puffer-frei-Interrupt ein
    UCSR1B |= (1<<UDRIE1);
}
```



Die Interruptroutine und die BADISR

Die Puffer-frei-ISR versendet zirkulares immer das nächste Zeichen:

```
ISR(USART1_UDRE_vect){           //Puffer-frei ISR
    UDR1 = LCD_dat[lcd_idx];      //schicke nächstes
    lcd_idx++;                    //Zeichen
    //nach dem letzten folgt das erste Zeichen
    if (lcd_idx >= 32) lcd_idx = 0;
}
```

Die BADISR erhöht das letzte Anzeigezeichens (unten rechts) »als Fehlerzähler«. Falls das zugehörige Zeichen nicht ».« bleibt, ist ein Interrupt aufgetreten, für den keine ISR einprogrammiert ist:

```
ISR(BADISR_vect){                //Fehlerzähler (letztes
    lcd_incErr(31);              //Zeichen) hochzählen
}
```



3. LCD-Treiber (comir_lcd)

Testbeispiel

Definition des Anzeigeformats für das nachfolgende Testbeispiel:



```
#define INITSTR "W0:.._x_W1:.._x_Zeit :...s_E:..."
//Zeichenpositionen für Ausgaben
#define LCP_W0T    3 //Restzeit Timer-Kanal 0
#define LCP_W0Z    6 //Zustand Timer-Kanal 0
#define LCP_W1T   11 //Restzeit Timer-Kanal 1
#define LCP_W1Z   14 //Zustand Timer-Kanal 1
#define LCP_ZEIT  21 //Zeit seit Programmstart in s
//Fehlerzähler BAD_ISR hat LCD-Position 31
```

Im Bild steht der Fehlerzähler Zeichen 31 für »BADISR« auf 3. Die Fehlerzähler der Zeichen 29 und 30 sind ungenutzt.



Initialisierungsteil des Testbeispiels

```
int main(void){
    uint8_t z0='0', z1='0'; //Ausgabezustand
    tmr_init();             //Treiber initial.
    lcd_init((uint8_t*)INITSTR);
    // nicht behandelte Interrupt ca. alle 8 s
    // für den Test des Fehlerzählers
    TCCR4B = 0b101;        //Timer 4, Normalmodus,
    TIMSK4 = 1<<TOIE4;    //VT 1024, Überlaufint. ein
    sei();                 //Interrupts global ein
```

Zur Nachbildung zählbarer Fehlfunktionen werden mit Timer 4 periodisch Interrupts erzeugt, für die es keine ISR gibt.





Endlosschleife zum Testbeispiel

```
while(1){
    if (!tmr_restzeit(0)){//wenn Kanal 0 abgelaufen
        tmr_start(31, 0);    //Kanal 0 mit 3,1 s init.
        lcd_disp_chr(z0, LCP_WOZ);
        z0 ^=1;              //'0'(0x30) <=> '1'(0x31)
    }
    if (!tmr_restzeit(1)){//wenn Kanal 1 abgelaufen
        tmr_start(17, 1);    //Kanal 1 mit 1,7 s init.
        lcd_disp_chr(z1, LCP_W1Z);
        z1 ^=1;              //'0'(0x30) <=> '1'(0x31)
    }
    //Zeitwerte immer aktualisieren
    lcd_disp_val(tmr_restzeit(0),LCP_WOT, 2);
    lcd_disp_val(tmr_restzeit(1),LCP_W1T, 2);
    lcd_disp_val(tmr_get()/10, LCP_ZEIT, 4);
} //Ende der Endlosschleife
```



3. LCD-Treiber (comir_lcd)

- LCD-Modul mit Y-Kabel an JD oben anstecken. JDX »gekreuzt (=)«.
- LCD an JD unten. LCD-Jumper-Stellungen siehe Bild.
- Projekt »F8-test_comir_lcd\test_comir_lcd« öffnen.
- Übersetzen. Start im Debugger . Continue .
- Ausgabe kontrollieren.





Treiber PC-Kommunikation (comir_pc)



Änderungen gegenüber dem Treiber »comsf_pc«

- Ersatz der Schrittfunktion für Empfang und Senden durch je eine ISR für Empfang und Senden.
- Zusätzliche Empfangs-Timeout-ISR mit Timer 3. Löscht im Empfangspuffer Bytes halb empfangener Nachrichten nach 0,1s. Fehlertoleranz gegenüber Fehlansteuerungen vom PC.
- Drei Interrupt-Freigaben am Ende der Initialisierung.
- Interrupt-Sperren für Sequenzen zum Lesen und Schreiben von EA-Daten in der Send- und Get-Funktion.
- Zusätzliche private Variable und Lesefunktion für einen Zähler für Sende- und Empfangsfehler:

```
uint8_t com_pc_err_ct;  
uint8_t com_pc_err();  
//Rückgabe 1 wenn (com_pc_err_ct=0) sonst 0
```

Wurde ergänzt für die Fehlersuche. Unterbrechungspunkte vor Anweisungen zur Erhöhung des Fehlerzählers.



Private Daten

```
uint8_t rmsg[COM_PC_RMSG_LEN]; //Empfangspuffer
uint8_t smsg[COM_PC_SMSG_LEN]; //Sendepuffer
uint8_t sidx, ridx;           //Pufferzeiger
uint8_t last_byte;           //letztes empfangene Byte
uint8_t com_pc_err_ct;       //Fehlerzähler
```

Die Puffergrößen sind im Header definiert, in diesem Projekt für beide Puffer 4 Byte. Für andere Projekte im Header anzupassen.



Erweiterung der Initialisierungsfunktion

```
void com_pc_init(){  
    ... //Initialisierung und Einschalten USART2  
    ... //Empfangs- und Sendepuffer leer  
    UCSR2B |= (1<<RXCIE2); //Empfangs-Interrupt ein  
    TCNT3 = 0; //Zähler 3 Rücksetzen  
    TCCR3B = 0; //Zähltakt aus  
    OCR3A = 12500; //Empf.-Timeout 100 ms  
    TIMSK3 |= 1<<OCIE3A; //Vergleichs-Int. A ein  
}
```

- Der Sende- (Puffer-frei-) Interrupt bleibt bis zur Übergabe zu versendender Daten gesperrt.
- Timer 3 wird beim Empfang eines Bytes neu gestartet.
- Wenn von einer unvollständig empfangenen Nachricht 100 ms kein weiteres Byte ankommt, löscht die Timer 3-ISR die bereits empfangene Bytes.



ISR für Empfang und Empfangs-Timeout

```
ISR(USART2_RX_vect){           //ISR Empfang
    last_byte = UDR2;          //Byte lesen
    if (ridx < COM_PC_RMSG_LEN){ //wenn Platz im
        rmsg[ridx] = last_byte; //Puffer, übernehmen
        ridx++;
        TCNT3 = 0;             //Rücksetzen Zähler 3
        TCCR3B = 0b11;        //Zähltakt clk/64 ein
    }
}

ISR(TIMER3_COMPA_vect){        //ISR Empfangs-Timeout
//wenn der Empfangspuffer noch nicht gefüllt ist
    if ((ridx>0) && (ridx <COM_PC_RMSG_LEN)){
        ridx = 0;             //Puffer löschen
        com_pc_err_ct++;      //Fehlerzähler erhöhen
    }
    TCCR3B = 0;              //Zähltakt aus
}
```



ISR für das Senden

```
ISR(USART2_UDRE_vect){           //Sendepuffer-frei-ISR
  if (sidx < COM_PC_SMSG_LEN){ //wenn Sendepuffer frei
    UDR2 = msg[sidx];           //und Senddaten, diese
    sidx++;                     //versenden und Index
  }                             //erhöhen
  else
    UCSR2B &= ~(1<<UDRIE2);    //Puffer-frei-Int. aus
}
```

Wenn keine zu versendenden Daten mehr anstehen, MUSS der Puffer-Frei-Interrupt deaktiviert werden. Sonst wird nach jedem Maschinenbefehl des Hauptprogramms die ISR eingeschoben. Erhebliche Reduzierung der nutzbaren Verarbeitungsleistung.



Get-Funktion

Unterbrechungsfreies Kopieren der gesamten Nachricht aus dem Empfangspuffer in einen Nachrichtenpuffer des Programms.

Vom Test, ob Daten da sind, bis Abschluss der Datenübergabe, sind Empfangs-Interrupts gesperrt.

```
uint8_t com_pc_get(uint8_t *msg){
    //Interruptfreigabe speichern
    uint8_t tmp = UCSR2B;
    //Empfangs-Interrupt aus
    UCSR2B &= ~(1<<RXCIE2);
    //wenn der Empfangs-Puffer nicht voll
    if (ridx < COM_PC_RMSG_LEN){
        //Interrupt-Freigabe wiederherstellen
        UCSR2B = tmp;
        return 0;        //Rücksp. mit "falsch"
    }
}
```



4. Treiber PC-Kommunikation (comir_pc)

```
...
//sonst Empfangsnachricht kopieren
for (ridx=0; ridx<COM_PC_RMSG_LEN;ridx++)
  msg[ridx] = rmsg[ridx];//
ridx = 0;          //Empfangspuffer leeren
//Interrupt-Freigabe wiederherstellen
UCSR2B = tmp;
return 1;         //Rücksprung mit "wahr"
}
```




Send-Funktion

Unterbrechungsfrei gesamte Nachricht aus dem Nachrichtenpuffer des Programms in den Sendepuffer des Treibers kopieren.

```
uint8_t com_pc_send(uint8_t *msg){
    uint8_t tmp = UCSR2B; //Int.-Freigabe speichern
    UCSR2B &= ~(1<<UDRIE2); //Puffer-frei-Interrupt aus
    //wenn der Sendepuffer nicht leer ist
    if (sidx < COM_PC_SMSG_LEN){
        UCSR2B = tmp; //Int.-Freigabe wiederherst.
        return 0; //leer, Rückgabe "falsch"
    }
    for (sidx=0; sidx<COM_PC_SMSG_LEN;sidx++){
        msg[sidx] =msg[sidx]; //sonst Nachricht übergeben
        //und Zeiger auf Nachrichtenanfang
    }
    sidx = 0; //Sendepuffer voll
    UCSR2B |= 1<<UDRIE2; //Interrupt-Freigabe ein
    return 1; //Rücksprung mit "wahr"
}
```



Letztes Byte und Fehlerbehandlung

Rückgabe letztes Byte. Gedacht für Sofort-Nachrichten wie »Fahrzeughalt«, wenn im Empfangspuffer noch unabgearbeitete Steuernachrichten stehen.

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

Abfrage, ob der Fehlerzähler ungleich null ist und löschen des Fehlerzählers:

```
uint8_t com_pc_err(){  
    if (com_pc_err_ct){ //wenn Fehlerzähler >0  
        com_pc_err_ct = 0; //Fehlerzähler löschen  
        return 1;          //Rückkehr mit 1 (wahr)  
    }                      //sonst  
    return 0;             //Rückkehr mit 0 (falsch)  
}
```



4. Treiber PC-Kommunikation (comir_pc)

Die beiden Funktionen »com_pc_last_byte()« und »com_pc_err()« benötigen keine Unterbrechungssperre, weil die kritischen Sequenzen genau ein Byte kopieren, d.h. je nur aus einem nicht unterbrechbaren Maschinenbefehl bestehen.



Treiber Ultraschallsensor (comir_sonar)



Der Treiber »comir_sonar«

Bereitstellung Sonar-Abstandswerte. Gegenüber »comsf_sonar«

- schaltet die Init-Funktion den Empfangs-Interrupt frei.
- ISR statt Schrittfunktion.

```
void sonar_init(){
    ...//Initialisierung USART1: 8N1, 9600 Baud
    UCSR1B|=(1<<RXEN1); //Empfänger ein
    DDRD   |= 1<<PD5;    //PD5 Ausgang
    PORTD  |= 1<<PD5;    //Sonar einschalten
    snr_state =0;        //Empfangsautomat initial.
    UCSR1B|= 1<<RXCIE1; //Empfangsinterrupt ein
}
```

Achtung: Der Sendeteil von USART1 wird vom LCD-Treiber genutzt.
Beim Einschalten von Empfänger und Empfangs-Interrupts nicht
Sender und Sende-Interrupt ausschalten.

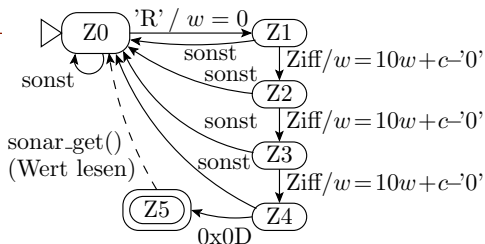


5. Treiber Ultraschallsensor (comir_sonar)

Aus der Schrittfunktion
wird die ISR

```

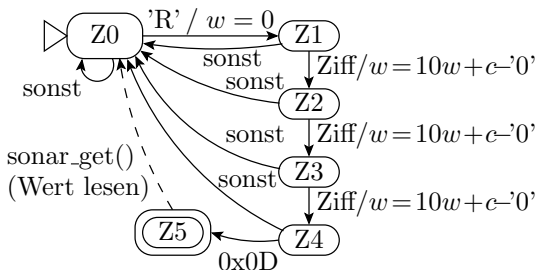
uint16_t snr_val;
uint8_t  snr_state;
ISR(USART1_RX_vect){
  uint8_t dat = UDR1;
  if (snr_state==0 && dat=='R'){
    snr_state = 1;           //Kante von Z0 nach Z1
    snr_val   = 0;
  } // "Ziff"-Kanten
  else if (snr_state>0 && snr_state<4
           && dat>='0' && dat<='9'){
    snr_val = (snr_val*10) + (dat-'0');
    snr_state++;
  } return;
  ...                       //Kante von Z4 nach Z5
}
    
```



Im Bild ist w »snr_val« und der Zustand »snr_state«.



5. Treiber Ultraschallsensor (comir_sonar)



```
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
}
```



Get-Funktion

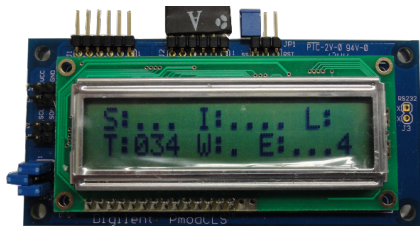
Die Get-Funktion muss während der Auswertung und Änderung des Automatenzustands und dem Lesen des Abstandswertes die Interrupt-Annahme verbieten:

```
uint8_t sonar_get(uint16_t *sptr){
    uint8_t tmp = UCSR1B; //Int.-Freigabe speichern
    UCSR1B &=~(1<<RXIE1); //Empfangs-Interrupt aus
    if (snr_state>4) { //wenn neuer Wert da,
        *sptr = snr_val; //ausgeben
        snr_state = 0; //Zustand rücksetzen
        UCSR1B = tmp; //Int.-Freigabe wiederherst.
        return 1; //Rücksprung mit "wahr"
    }
    UCSR1B = tmp; //Int.-Freigabe wiederherst.
    return 0; //Rücksprung mit "falsch"
}
```




Testbeispiel mit allen Treibern

Testbeispiel mit allen Treibern



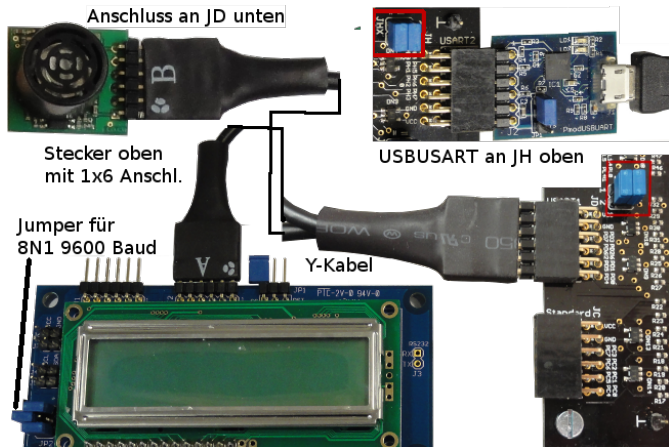
Ausgabe nach Programmstart:

- S: Sonar-Abstandswert (Treiber »comir_sonar«)
- I: 4-Zeichen-Nachricht vom PC (Treiber »comir_pc«)
- L: letztes vom PC empfangenes Byte (Treiber »comir_pc«)
- T: Zeit seit Programmstart in s (Treiber »comir_timer«)
- W: Wartezeit bis zur Ausgabe (Treiber »comir_timer«).
- E: Fehlerzähler (Treiber »comir_lcd«)

- Programm wartet auf eine mindestens 4 Bytes lange Nachricht.
- Wartet dann 8 s mit Count-Down-Anzeige.
- Schickt dann den Sonar- und zwei Zählwerte zurück.



6. Testbeispiel mit allen Treibern



- »PmodMAXSONAR« und »PModPTH2« wie im Bild über Y-Kabel an JD (LCD oben, Sonar unten).
- PModUSBUSART an JH oben und über USB zum PC.
- Jumper JHK und JDJ »gekreuzt (=)«.



- Projekt »F8-comir\comir« öffnen, übersetzen und Programm starten.
- HTerm starten, 8N1 9600 Baud, Connect.
- Zeichenfolge »asdf« schicken.
- Kontrolle Nachrichtanzeige LCD.
- Weitere Einzelzeichen schicken und Kontrolle »letztes empfangenes Byte«.
- Countdown von 8 auf 0 s und HTerm-Zeichenempfang abwarten. Kontrolle des im HTerm empfangenen 2-Byte-Sonarwerts (in Zoll), des 1-Byte-Sonar-Zählerwerts und des 1-Byte Nachrichtenzählers auf Plausibilität.



```
S:... I:asdf L:f
T:073 W:0 E:..18
```

Fehlerzähler:

- 1 Versendefehler (nicht künstlich erzeugt).
- 2 Empfangs-Timout: Inkrement bei Nachrichtenlänge \neq 4.
- 3 Testzähler: Inkrement bei jedem Nachrichtenempfang.
- 4 BADIR-Zähler: Inkrement ca. alle 8 s (Tmr4-Überlauf).

Konstanten für die Anzeige



```
S:... I:asdf L:f
T:073 W:0 E:..18
```

```
#define INITSTR "S:... I:... L:. T:... W:.. E:.... "
#define LCP_SONAR      2 //Sonarwert
#define LCP_RMSG       8 //Eingabedaten
#define LCP_LBYTE      15 //letztes empf. Byte
#define LCP_TIME       18 //Zeitanzeige
#define LCP_WAIT       24 //Wartezeit
//Anzeigepositionen für Fehlerzähler
#define ERR_SEND       28 //Sendeversagen
#define ERR_ETO        29 //Empfangs-Timeout
#define ERR_TEST       30 //zählt Empfangsnachrichten
//Zeichen 31 ist der Zähler falsche Interrupts
```

Initialisierung

```
int main(void){
    uint8_t state='E';    //Programmzustand {E,V,A}
    uint16_t snrval;     //Sonarwert
    uint8_t sn_ct,msg_ct;//Sonarwert,Nachrichtenzähler
    sonar_init();       //alle Hintergrundprozesse
    com_pc_init();      //initialisieren
    lcd_init((uint8_t*)INITSTR);
    tmr_init();
    //nicht behandelte Interrupt ca. alle 8 s
    TCCR4B = 0b101;     //Tmr 4, Normalmodus,
    TIMSK4 = 1<<TOIE4; //VT 1024, Überlaufs-Int. ein
    sei();              //Interrupts global freigeben
    while(1) {
        ...
    }
}
```

In der Hauptschleife

```
if (sonar_get(&snrval)){//wenn neue Sonardaten
    //Sonarwert auf LCD ausgeben
    lcd_disp_val(snrval, LCP_SONAR, 3);
    sn_ct++;           //Sonarwerte zählen
}
if (state=='E'){           //wenn Zustand "Eingabe"
    if (com_pc_get(mrmsg)){//Wenn neue PC-Nachricht
        //diese übernehmen, auf LCD ausgeben
        lcd_disp_str(mrmsg, LCP_RMSG, COM_PC_RMSG_LEN);
        lcd_incErr(LCP_TESTERR);//Testfehlerzähler ++
        msg_ct++;
        tmr_start(80, 0); //Tmr-Kanal 0 für 8s starten
        state = 'V';      //Folgezustand "Verarbeitung"
    }
}
... //Fortsetzung nächste Folie
```

Hauptschleife Fortsetzung 1

```
else if (state=='V'){//wenn Zustand "Verarbeiten"  
    //Ausgabe Wartezeit bis zur nächsten EA-Operation  
    lcd_disp_val(tmr_restzeit(0)/10, LCP_WAIT, 1);  
    if (!tmr_restzeit(0))  
        state = 'A';           //Folgezustand "Ausgabe"  
}  
else{           //wenn Zustand "Ausgabe"  
    msmsg[0] = snrval >> 8; //Sensor- und Zählwerte  
    msmsg[1] = snrval & 0xFF; //byteweise in die Send-  
    msmsg[2] = sn_ct;         //nachricht schreiben  
    msmsg[3] = msg_ct;  
    if (!com_pc_send(msmsg)) // "string" versenden, wenn  
        lcd_incErr(ERR_SEND); //erfolglos, Fehlerz. ++  
    state = 'E';           //Folgezustand "Eingabe"  
}  
//immer letztes empf. Byte auf LCD schreiben  
    lcd_disp_chr(com_pc_last_byte(), LCP_LBYTE);
```


Hauptschleife Fortsetzung 2

```
//immer Zeit seit Programmstart in s ausgeben
lcd_disp_val((tmr_get()/10) % 1000, LCP_TIME, 3);
if (com_pc_err())           //Wenn Empfangs-Timeout
    lcd_incErr(ERR_ET0);    //Fehlerzähler erhöhen
}                            //Ende der Hauptschleife
```

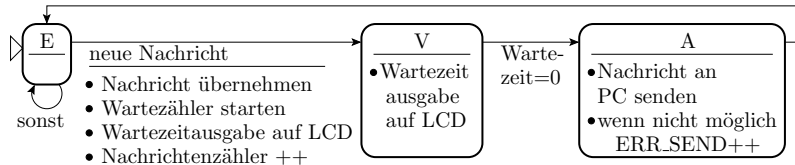
Kontrollfragen zum Testprogramm:



- 1 Beschreiben Sie den Zustandsablauf des Testprogramms durch einen Automatengraphen.
- 2 Was passiert, wenn der PC die nächste 4-Byte-Nachricht sendet, bevor der 8-s-Timeout abgelaufen ist?
- 3 Wird der Timeout-Ausgabewert im Programmzustand 'A' (Ausgabe) gelöscht?
- 4 Wie könnte man zum Testen einen Sendefehler provozieren?

Lösung

1 Zustandsgraph des Testprogramms:



- Die nächste während der 8 s-Wartezeit vom PC gesendete Nachricht wird vom Treiber angenommen, weitere nicht.
- Restzeitähler und -ausgabe sind im Zustand »A« null und brauchen deshalb nicht gelöscht zu werden.
- Ein Sendefehler lässt sich provozieren, indem pro Paket nicht genau vier Zeichen gesendet werden.

Zusammenfassung

Behandelt wurden die Treiber:

- `comir_tmr`: Bereitstellung von 4 Wartefunktionen für nebenläufige Aktivitäten und eine Systemuhr.
- `comir_lcd`: Bereitstellung von Anzeigefunktionen für das LCD, vorgesehen für Test- und Statusausgaben des zu entwickelnden Fahrzeuges.
- `comir_pc`: Bereitstellung einer Sende- und einer Empfangsfunktion für Datenpakete mit einer bei der Übersetzung festzulegenden Größe.
- `comir_sonar`: Bereitstellung sonarer Abstandswert.

Auf nachfolgenden Foliensätze werden weitere Treiber bzw. Testprogramme, aus denen Treiber zu entwickeln sind, behandelt:

- Motorensteuerung, Wegemessung, Motorregelung,
- Joystick, IR-Abstandssensor und Bodensensor.



Aufgaben



Aufgabe 8.1: Unabhängige LED-Blinksequenzen

Schreiben Sie unter Nutzung der 4 Kanäle des Treibers »comir_tmr« ein Programm, das die folgenden LEDs an Port J mit nachfolgenden Periodendauern blinken lässt:

LED	0	1	2	3
Periode	0,6s	1,4s	2,1s	3,4s

Hinweis: Das Hauptprogramm hat in der Endlosschleife folgende Struktur:

- wenn Timer-Kanal 0 abgelaufen, invertiere LED0 und starte Timer-Kanal 0 erneut mit ...
- wenn Timer-Kanal 1 abgelaufen, invertiere LED1 und starte Timer-Kanal 1 erneut mit ..., etc.



Aufgabe 8.2: Gepulste Signalausgabe

Schreiben Sie unter Verwendung der Treiber »comir_tmr« und comir_pc« ein Program, das in der Endlosschleife auf ein Byte vom PC wartet und nach Empfang dessen Wert als Blinksequenz auf LED0 beginnend mit Bit 0 wie folgt ausgibt:

- Bitwert 0: 0,2 s leuchten und 0,4 s aus,
- Bitwert 1: 0,4 s leuchten und 0,2 s aus.



Aufgabe 8.3: Test mit Logikanalysator

Visualisierung der Zeitverläufe im Testprogramm »test_comir« (ab Folie 42) mit dem USB-LOGI.

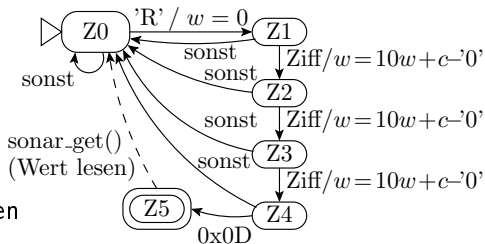
- Anschluss CH0 bis CH7 des USB-Logi an JA (PA0 bis PA7).
- Testprogramm und Treiber so ändern, dass jede ISR und jedes Unterprogramm eine Nummer ≥ 2 ausgibt und in »main« für den Aufzeichnungsbeginn das aufzuzeichnende Byte von 0b0000 0000 nach 0b0000 0001 wechselt.
- Aufzeichnungstaktperiode ca. 1 μ s (5 bis 10 Maschinenbefehle).

Aufgabe 8.4: Sonar-Treiber umschreiben

Der Treiber »comir_sonar« übernimmt immer erst den nächsten Sonar-Wert, wenn der vorhergehende mit »get_sonar()« abgeholt ist. Ändern Sie die ISR des Treibers so, dass »get_sonar()« immer den zuletzt vollständig übertragenen Wert zurückgibt.

Hinweise:

- Getrennte private Treibervariablen für den Zwischenwert »w« und den Ergebniswert.
- Änderung des Ablaufgraphen so, dass von Z4 bei Empfang von »0x13« nach Z0 übergegangen und das Ergebnis in die Ergebnisvariable kopiert wird. Der Zustand Z5 entfällt.





Aufgabe 8.5: Bluetooth-Treiber

- Entwickeln Sie in Anlehnung an »comir_pc« einen Treiber für die Kommunikation über Bluetooth mit dem PC (oder ihrem Handy).
- Testen Sie diesen, indem Sie die Treiber für die PC-Kommunikation durch den Bluetooth-Treiber ersetzen.
- Probieren Sie, ob Sie auf ihrem Handy eine App zum laufen bekommen, die über Bluetooth Daten mit dem PC austauschen kann.



Aufgabe 8.6: Benutzung der Comir-Treiber

Die besprochenen Treiber und den selbst zu entwickelnden Bluetooth-Treiber können Sie in ihrem eigenen Projekt nutzen.

- Denken Sie sich ein Steuerprogramm für ihr Fahrzeug aus.
- Stellen Sie die geplanten Motoraktivitäten auf dem LC-Display dar.
- Simulieren Sie externe Ereignisse (Weg abgefahren, Hindernis erkannt, ...) mit Timern und LED-Ausgaben.