

Informatikwerkstatt, Foliensatz 10 Interrupt-basierte Treiber

G. Kemnitz

7. Dezember 2020

Inhalt:

Inhaltsverzeichnis

1	Timer-Treiber (comir_tmr)	1
2	LCD-Treiber (comir_lcd)	4
3	Treiber PC-Kommunikation (comir_pc)	7
4	Treiber Ultraschallsensor (comir_sonar)	10
5	Testbeispiel mit allen Treibern	12
6	Aufgaben	16

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)

1 Timer-Treiber (comir_tmr)

Der Treiber »comir_tmr«

Der Timer-Treiber stellt eine Systemuhr und vier Timer bereit. Damit sollen Abläufe gesteuert werden, wie:

- Fahre maximal 10 s geradeaus,
- wenn der Sensorwert für 1 s größer 25, dann breche Fahrt ab, ...

Öffentliche Funktionen:

1. Lesen der Zeit seit Programmstart in 100 ms-Schritten:

```
uint32_t tmr_get();
```

2. Start eines Timers:

```
void tmr_start(uint16_t tw, uint8_t nr);
```

(tw – Zeit in 100 ms-Schritten; nr ∈ {0, 1, 2, 3}– Timer-Nr.).

3. Lesen der Restzeit in 100 ms-Schritten:

```
uint16_t tmr_restzeit(uint8_t nr);
```

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 Vergleichregister, 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 Programmzeitähler und dekrementiert die Wartezeitähler 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 Wartezeit.
        if (tmr_array[idx]) //wenn ungleich null
            tmr_array[idx]--; //abwärts zählen
}
```

Zugriffsmethoden für das Anwenderprogramm

```
uint32_t tmr_get(){ //Zeitähler 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? Werden Daten bearbeitet, die die ISR möglicherweise verändert?

- 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.
}

uint16_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;
}

uint32_t tmr_get(){        //Zeitähler 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 alle 0,7s und Timer-Kanal 1 soll die LED an PJ6 alle 1,2s invertieren.
- Auf den LEDs an PJ0 bis PJ4 soll der Zeitwert in Sekunden ausgegeben werden.

-
- Projekt »F10-3_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

```

- 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

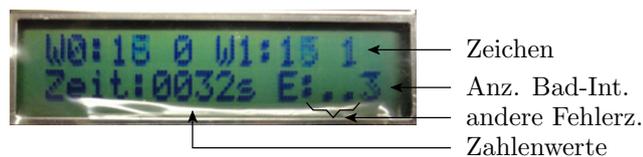
```

Anregungen zum Experimentieren:

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

2 LCD-Treiber (comir_lcd)

Der Treiber »comir_lcd«



Das LC-Display ist zur Statusausgabe vorgesehen:

- Programmzustände,
- Sensorwerte,
- Eingaben,
- Fehlerzähler, ...

Die ISR als Ersatz der Schrittfunktion sendet zeichenweise zyklisch einen als private Daten gespeicherten Text an das LC-Display.

Dieselben öffentlichen Funktionen wie Treiber mit Schrittfunktion:

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

Änderungen zur Umstellung auf Interrupts:

- Aktivierung Sendepuffer-frei-Interrupt am Ende der Initialisierungsfunktion.
- ISR statt Schrittfunktion.
- BADISR mit dem letzten Anzeigezeichen als Fehlerzähler.

Dieselben privaten Daten:

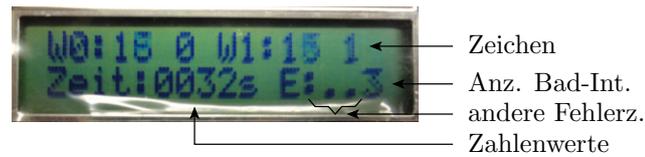
```
uint8_t LCD_dat[32]; //Ausgabertext
uint8_t lcd_idx;    //Indexvariable
```

Sendepuffer-frei-Interrupt ein am Ende der Initialisierung:

```
void lcd_init(uint8_t *text){//LCD-Treiber init.
    ... //wie in comsf USART1 8N1, Sender ein
    ... //LC-Display initialisieren
    ... // Init-Text in LCD_dat[32] kopieren
    UCSR1B |= (1<<UDRIE1);
}
```

Puffer-Frei-ISR: Zirkulares Versenden des Pufferinhalts:

```
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 Bad-ISR zur Erhöhung des letzte Anzeigezeichen (unten rechts) »als Fehlerzähler«:

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

Falls das zugehörige Zeichen wie im Bild nicht ».« bleibt, sind unbehandelte Interrupts aufgetreten (Programmierfehler).

Funktionstest

Definition des Anzeigeformats für das nachfolgende Testbeispiel:



```
#define INITSTR "W0:.._x_W1:.._x_Zeit:....s_E:..."
// Zeichenpositionen für Ausgaben
#define LCP_WOT 3 // Restzeit Timer-Kanal 0
#define LCP_WOZ 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
#define LCP_FCT1 29 // Fehlerzähler (ungenutzt)
#define LCP_FCT2 30 // Fehlerzähler (ungenutzt)
#define LCP_BISR 31 // Fehlerzähler Bad-ISR
```

Initialisierungsteil Funktionstest

```
int main(void){
    uint8_t z0='0', z1='0'; //Ausgabezustand
    tmr_init();             //Treiber initial.
    lcd_init((uint8_t*)INITSTR);
    // nicht behandelter Interrupt ca. alle 8 s
    TCCR4B = 0b101;        //Tmr4, Normalmod.,  $f_{ct} = \frac{8\text{MHz}}{1024}$ 
    TIMSK4 = 1<<TOIE4;    //Freigabe Überlaufinterrupt
    sei();                  //Interrupts global ein
```

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

Endlosschleife

```

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_W0T, 2);
  lcd_disp_val(tmr_restzeit(1),LCP_W1T, 2);
  lcd_disp_val(tmr_get()/10, LCP_ZEIT, 4);
} //Ende der Endlosschleife

```

Ausprobieren

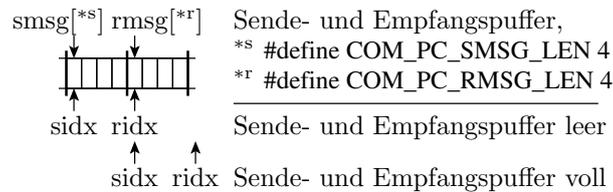
- LCD-Modul mit Y-Kabel an JD oben anstecken. JDX »gekreuzt (=)«.
- LCD an JD unten. LCD-Jumper-Stellungen siehe Bild.
- Projekt »F10-test_comir_lcd\test_comir_lcd« öffnen.
- Übersetzen. Start im Debugger . Continue .
- Ausgabe kontrollieren.



3 Treiber PC-Kommunikation (comir_pc)

Der Treiber für die PC-Kommunikation (comir_pc)

- Nutzung Sender und Empfänger von USART2.
- Erwartet Modul PmodUSBUSART an Port H.
- Private Daten und Funktionsweise:



Initialisierungs-, Sende- und Empfangsfunktion wie »comsf_pc«:

```

void com_pc_init(); //Initialisierung
uint8_t com_pc_get(uint8_t *msg); //Empfang
uint8_t com_pc_send(uint8_t *msg); //Senden
uint8_t com_pc_last_byte(); //letztes empfangenes Byte
  
```

ISR statt Schrittfunktion

Ersatz der Schrittfunktion durch je eine ISR für Empfang und Senden:

```

ISR(USART2_RX_vect){ //ISR Empfang
  last_byte = UDR2; //Byte lesen
  if (ridx < COM_PC_RMSG_LEN){//wenn Pufferplatz frei
    rmsg[ridx] = last_byte; //Byte in Puffer
    ridx++; //Index erhöhen
  }
}
ISR(USART2_UDRE_vect){ //Sendepuffer-frei-ISR
  if (sidx < COM_PC_SMSG_LEN){//wenn Sendepuffer leer
    UDR2 = smsg[sidx]; //nächsten Wert senden
    sidx++; //Sendeindex erhöhen
  }
  else { //sonst
    UCSR2B &= ~(1<<UDRIE2); //Puffer-frei-Int. aus
  }
}
  
```

Weitere ISR für Empfangs-Timeout

Wenn Empfangsdaten unvollständig und Timer 3 Vergleichswert »OCR3A« erreicht, löschen der bisher empfangenen Daten:

```

ISR(TIMER3_COMPA_vect){ //ISR Empfangs-Timeout
  if ((ridx>0) && (ridx < COM_PC_RMSG_LEN)){
    ridx = 0; //Puffer löschen
    ...; //Fehlerzähler erhöhen
  }
  TCCR3B = 0; //Zähltakt aus
}
  
```

Erweiterung der Empfangs-ISR:

```

ISR(USART2_RX_vect){ //ISR Empfang
  < wenn Puffplatz frei Datenübernahme >
  if (ridx < COM_PC_RMSG_LEN){//wenn Empf. unvollständig
    TCNT3 = 0; //Rücksetzen Zähler 3
    TCCR3B = 0b11; //Zähltakt clk/64 ein
    TIMSK3 |= 1<<OCIE3A; //Vergleichs-Int. A ein
  }
  else TIMSK3 &= ~(1<<OCIE3A); //sonst Vergleichs-Int. A aus
}
  
```

Private Daten:

```
#define COM_PC_RMSG_LEN 4      // Empfangspuffergröße
#define COM_PC_SMSG_LEN 4      // Sendepuffergröße
uint8_t rmsg[COM_PC_RMSG_LEN]; // Empfangspuffer
uint8_t smsg[COM_PC_SMSG_LEN]; // Sendepuffer
uint8_t sidx, ridx;           // Puffer-Index
uint8_t last_byte;           // letztes empf. Byte
uint8_t com_pc_err_ct;       // Fehlerzähler*1
```

(*1 alternativ Nutzung Fehlerzähler LCD-Treiber). Init.-Funktion:

```
void com_pc_init(){
    ... //Initialisierung und Einschalten USART2
    ridx = 0; sidx = 0; // Empfang- und Sendepuff. leer
    UCSR2B |= (1<<RXIE2); // Empfangs-Interrupt ein*2
    TCNT3 = 0; // Zähler 3 zurücksetzen
    TCCR3B = 0; // Zähltakt aus
} OCR3A = 12500; // Empf.-Timeout 100 ms
```

*2 Der Sendepuffer-frei-Interrupt darf nur eingeschaltet sein, wenn Daten zum Senden da sind. Sonst startet nach jedem Maschinenbefehl die Puffer-frei-ISR und verbraucht > 90% der Rechenleistung.

ISR-Sperren in öffentlichen Funktionen

Öffentliche Funktionen, die private ISR-Daten mit mehr als einem Maschinenbefehl bearbeiten, darf die ISR nicht unterbrechen (Gefahr ungewollter Datenveränderungen).

```
uint8_t com_pc_get(uint8_t *msg){
    uint8_t UCSR2B_bak = UCSR2B; //ISR-Freigabe speichern
    UCSR2B &= ~(1<<RXIE2); //Empfangs-Interrupt aus
    if (ridx >= 4){ //wenn der Puffer voll
        for (ridx=0; ridx<4;ridx++) msg[ridx] = rmsg[ridx];
        ridx = 0; //Empfangspuffer leeren
        UCSR2B = UCSR2B_bak; //ISR-Freigabe rücksetzen
        return 1; //Rückspr. "Daten übergeben"
    }
    UCSR2B = UCSR2B_bak; //ISR-Freigabe rücksetzen
    return 0; //Rückspr. "keine Daten"
}
```

ISR-Sperre erforderlich ab Test, ob Daten da, bis Datenübergabe.

Send-Funktion:

```
uint8_t com_pc_send(uint8_t *msg){
    uint8_t UCSR2B_bak =UCSR2B; //Int.-Freig. speichern
    UCSR2B &= ~(1<<UDRIE2); //Puffer-frei-Int. sperren
    if (sidx >= 4){ //wenn Puffer versendet
        for (sidx=0; sidx<4;sidx++)
            msg[sidx] =smsg[sidx]; //Nachricht übergeben
        sidx = 0; //Sendeindex auf 1. Zeichen
    }
```

```

    UCSR2B = UCSR2B_bak; //Int.-Freig. rücksetzen
    return 1;           //Rückspr. "Dat. übernomm."
}
    UCSR2B = UCSR2B_bak; //Int.t-Freig. rücksetzen
    return 0;           //Rücksprung ohne Daten-
}                       //           übernahme
}

```

- Eine Unterbrechung durch die Puffer-frei-ISR an einer zufälligen Stelle würde zu einem inkonsistenten Treiberzustand führen.
- ISR am Ende von *jedem* Abarbeitungspfad wieder freigeben.

Rückgabe »letztes Byte«: Zugriff nur mit einem Maschinenbefehl auf private Daten (ein Byte lesen). ISR-Sperre nicht zwingend:

```

uint8_t com_pc_last_byte(){
    return last_byte;
}

```

Fehlerzähler lesen und löschen: Zugriff mit mehr als einen (Maschinen-) Befehl auf private Daten. ISR-Sperre notwendig:

```

uint8_t com_pc_err(){
    uint8_t UCSR2B_bak = UCSR2B; //Freig. speichern.
    UCSR2B &= ~(1<<UDRIE2); //Puffer-frei-Int. aus
    if (com_pc_err_ct){ //wenn Fehlerzähler >0
        com_pc_err_ct = 0; //Fehlerzähler löschen
        UCSR2B = UCSR2B_bak; //Int.t-Freig. rücksetzen
        return 1;           //Rückkehr mit 1 (wahr)
    }                       //sonst
    UCSR2B = UCSR2B_bak; //Int.t-Freig. rücksetzen
    return 0;           //Rückkehr mit 0 (falsch)
}

```

4 Treiber Ultraschallsensor (comir_sonar)

Der Treiber »comir_sonar«

Bereitstellung Sonar-Abstandswerte.

- Gegenüber »comsf_sonar« schaltet die Init-Funktion den Empfangs-Interrupt frei:

```

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; //Empfangs-Interrupt ein
}

```

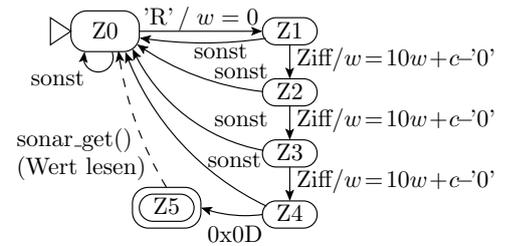
- Achtung: Der Sendeteil von USART1 wird vom LCD-Treiber genutzt. Beim Einschalten von Empfänger und Empfangs-Interrupt Sender und Sende-Interrupt anlassen (»|=« statt »=«).

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++;
    } ...                      //Kante von Z4 nach Z5
}

```



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

```

    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 mit ISR-Sperre

```

uint8_t sonar_get(uint16_t *sptr){
    uint8_t UCSR1B_bak = UCSR1B; //Int.-Freig. speich.
    UCSR1B &=~(1<<RXCIE1); //Empfangs-Interrupt aus
    if (snr_state>4) { //wenn neuer Wert da,
        *sptr = snr_val; //ausgeben
        snr_state = 0; //Zustand zurücksetzen
        UCSR1B = UCSR1B_bak; //Int.-Freigabe wiederherst.
        return 1; //Rückspr. "Daten übergeben"
    }
    UCSR1B = UCSR1B_bak; //Int.-Freigabe wiederherst.
    return 0; //Rückspr. "keine Übergabe"
}

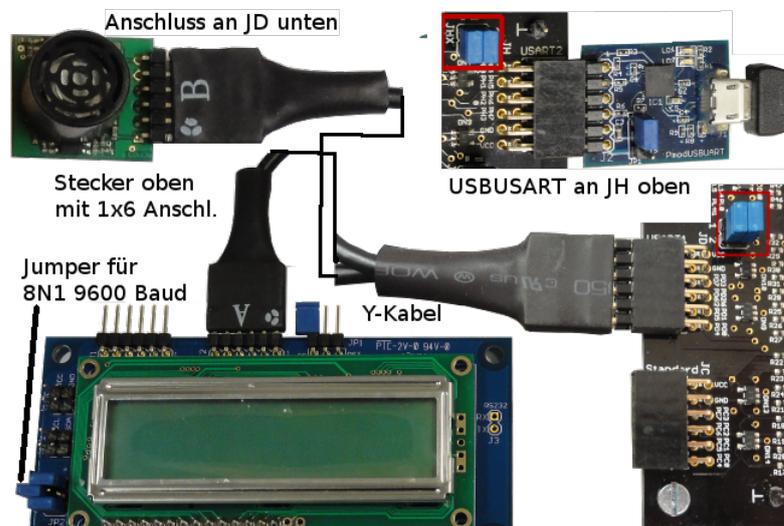
```

- Der Treiber beginnt erst mit dem Empfang eines neuen Wertes, nach dem der zuvor empfangene Sonarwert abgeholt ist.
- Aufgabe zum selber lösen: Treiber so umschreiben, dass auch dann der letzte vom Sensor übertragene Wert zurückgegeben wird, wenn zwischenzeitlich nicht alle Werte abgeholt werden.

5 Testbeispiel mit allen Treibern

Schaltung vorbereiten

- Spannung ausschalten.
- Jumper JHX und JDX »gekreuzt (=)«.
- PModUSBUSART an MySys JH oben.
- PModUSBUSART mit PC verbinden.
- Y-Kabel Doppel MySys JD.
- Y-Kabel Einzel oben (A) an LCD J2 .
- Auf LCD Jumper JP2 MD0 und MD2 (8N1, 9600 Baud).
- Y-Kabel Einzel unten (B) an PmodMAXSONAR.
- Spannung einschalten.



- Projekt »F10-comir\comir« öffnen, übersetzen und starten.
- HTerm starten.

Ausgaben 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 4 Byte-Nachricht.
- Wartet dann 8 s mit Count-Down-Anzeige.
- Schickt dann den Sonar- und zwei Zählwerte zurück.

Ausprobieren

- HTerm 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.

Fehlerzähler:

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

Walk-Through durch das Testprogramm

Konstanten für die Anzeigen:

```
#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-Timout
#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 behandelter Interrupt ca. alle 8 s
  TCCR4B = 0b101; //Tmr 4, Normalmodus,
  TIMSK4 = 1<<TOIE4; //VT 1024, Überlaufs-Int. ein
  sei(); //Interrupts global freigeben
  while(1) {
    ...
  }
}
```

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 Zust. "Eingabe"
      if (com_pc_get(mrmmsg)){//wenn neue PC-Nachr.
        //diese übernehmen, auf LCD ausgeben
        lcd_disp_str(mrmmsg, LCP_RMSG, COM_PC_RMSG_LEN);
        lcd_incErr(LCP_TESTERR);//Testfehlerzähler ++
        msg_ct++;
        tmr_start(80, 0); //Tmr-Kanal 0 Start für 8s
        state = 'V'; //Folgezust. "Verarbeitung"
      }
    }
    else if (state=='V'){//wenn Zust. "Verarbeiten"
      //Ausg. Wartezeit bis zur nächsten EA-Operation
      ... //Fortsetzung nächste Folie

  lcd_disp_val(tmr_restzeit(0)/10, LCP_WAIT, 1);
  if (!tmr_restzeit(0))
    state = 'A'; //Folgezustand "Ausgabe"
}
else{ //wenn Zustand "Ausgabe"
  msmmsg[0] = snrval >> 8; //Sensor- und Zählwerte
  msmmsg[1] = snrval & 0xFF;//byteweise in die Send-
  msmmsg[2] = sn_ct; //nachricht schreiben
  msmmsg[3] = msg_ct;
  if (!com_pc_send(msmmsg)) //"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);
//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
}

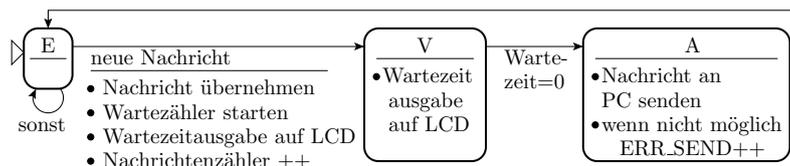
```

Fragen 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:



2. Die nächste während der 8 s-Wartezeit vom PC gesendete Nachricht wird vom Treiber angenommen, weitere nicht.
3. Restzeitähler und -ausgabe sind im Zustand »A« null und brauchen deshalb nicht gelöscht zu werden.
4. 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.

6 Aufgaben

Aufgabe 10.1: Testbeispiele aus der Vorlesung

Ausprobieren der Testbeispiele für

- den Timer-Treiber,
- den LCD-Treiber und
- für alle Treiber zusammen.

Aufgabe 10.2: 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	1,4 s	2,1 s	3,4 s

Hinweis: Das Hauptprogramm hat in der Endlosschleife folgende Struktur:

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

Aufgabe 10.3: Gepulste Signalausgabe

Schreiben Sie unter Verwendung der Treiber »comir_tmr« und comir_pc« ein Programm, das in der Endlosschleife auf ein Byte vom PC wartet und nach Empfang dessen Wert als Blinksequenz auf LED 0 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 10.4: Test mit Logikanalysator

Visualisierung der Zeitverläufe im Testprogramm »test_comir« (ab Folie 12) mit dem USB-LOGI.

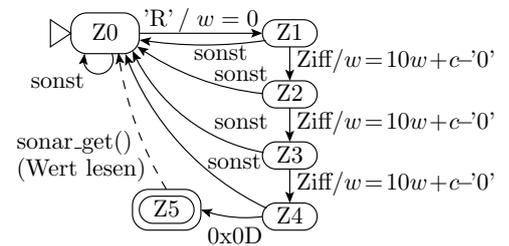
- Anschluss CH0 bis CH7 des USB-Logi an JA (PA0 bis PA7).
- Testprogramm und Treiber so ändern, dass jede ISR und jedes Unterprogramm eine Nummer ≥ 2 ausgibt und in »main« für den Aufzeichnungsbeginn das aufzuzeichnende Byte von 0b0000 0000 nach 0b0000 0001 wechselt.
- Aufzeichnungstaktperiode ca. 1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

Aufgabe 10.5: 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 »0x0D« nach Z0 übergegangen und das Ergebnis in die Ergebnisvariable kopiert wird. Der Zustand Z5 entfällt.

**Aufgabe 10.6: 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 10.7: 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.