Informatikwerkstatt, Foliensatz 7 Interrupts, LCD-Zustandsmonitor, Steuerschrittfunktion

G. Kemnitz

Institut für Informatik, Technische Universität Clausthal

9. Januar 2014

Inhalt des Foliensatzes

Interrupts

LCD-Zustandsmonitor

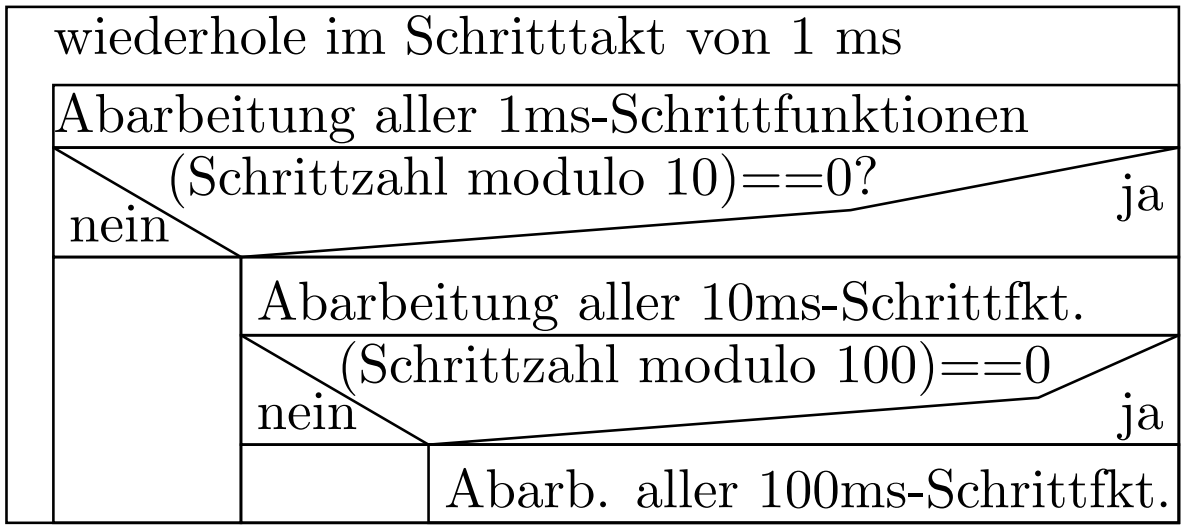
Erweiterung um die Abstandsmessung

Interrupts

Aufgaben mit unterschiedlicher Schrittzeit

- Steuerung der Fahrzeugs: 100ms bis mehrere Sekunden.
- Reaktion auf Sensordaten, Regelkreise: typ. 10 ms
- Serielle Kommunikation, Sensorabfrage: < 1 ms.

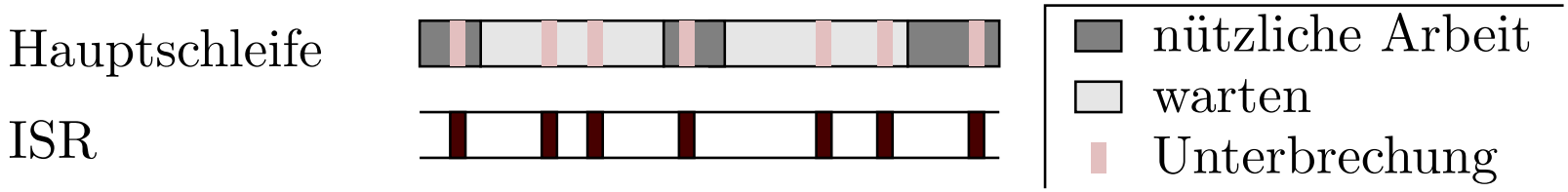
Lösung ohne Interrupts:



In Summe dürfen die Bearbeitungszeiten aller Schrittfunktionen zusammen nicht länger als die kürzeste Schrittzeit sein.

Interrupts

Hardware-Funktionen zur Unterbrechung des laufenden Programms durch Tasks mit kurzen Schrittzeiten:



Ein Interrupt (Unterbrechung):

- muss explizit freigeschaltet werden,
- wird dann durch Aktivierung eines Ereignisbits ausgelöst und
- bewirkt einen Sprung zum Interruptvektor (hardware-codierte Programmadresse).
- Ab dieser Adresse steht die ISR (Interrupt Service Routine).
- Eine ISR endet mit einem Rücksprung zum unterbrochenen Programm und ist in der Regel selbst nicht unterbrechbar.

Im weiteren genutzte Interrupts

- OCF3A, OCF3B: Timer3 Vergleichs-Interrupt (für die nachfolgenden Experimente).
- UDRE0: USART0 Sendepuffer frei (Ausgabe an den PC).
- RXC0: USART0 Datenempfang (Eingabe vom PC).
- UDRE1: USART1 Sendepuffer frei (LCD-Ausgabe).
- SPIF: SPI Transfer beendet (Joystick Ein- und Ausgabe).
- ADIF: Analog-Digital-Wandlung beendet (Abstandssensor).

Jeder Interrupt hat ein Freigabebit und einen Interruptvektor:

Int.-Bit (Register)	Freigabebit (Register)	Interrupt-Vektor
OCF3A (ETIFR)	OCIE3A (ETIMSK)	TIMER3_COMPA_vect
UDRE0 (UCSR0A)	UDRIE0 (UCSR0B)	USART0_UDRE_vect
RXC0 (UCSR0A)	RXCIE0 (UCSR0B)	USART0_RX_vect
UDRE1 (UCSR1A)	UDRIE1 (UCSR1B)	USART0_UDRE_vect

Int.-Bit (Register)	Freigabebit (Register)	Interrupt-Vektor
SPIF (SPSR)	SPIE (SPCR)	SPI_STC_vect
ADIF (ADCSRA)	ADIE (ADCSRA)	ADC_vect

Zur Nutzung von Interrupts ist

- der Header avr/interrupt.h einzubinden
- eine ISR zu definieren[1]:

```
ISR(<Interrupt-Vektor>){...}
```

- der Interrupt lokal freizugeben, z.B. für OCF3A

```
ETIMSK |=1<<OCIE3A;
```

- und es sind Interrupts global freizugeben:

```
sei()
```

[1] Die Definition einer ISR kann in einer beliebigen c-Datei des Projekts erfolgen. Eine ISR ist keine normale Funktion, sondern eine, die mit goto <interrupt-Vektor> aufgerufen wird, keine Übergabeparameter und keinen Rückgabewerte hat.

Risiken von Interrupts

Interrupts können, wenn freigegeben, ein Programm an beliebiger Stelle unterbrechen. Das ermöglicht viele zusätzliche Programmierfehler, die zum Teil sehr schwer zu finden sind:

- Für ungenutzte Interrupts programmiert der Compiler einen Systemneustart. Vermeidbar durch Definition einer alternativen ISR für ungenutzte Interrupts:

  ```
  ISR(BADISR_vect);
  ```

- Wenn das unterbrochene Programm gerade ein Datenobjekt bearbeitet, das die ISR verwendet, entsteht Datenchaos. Deshalb sind während der Abarbeitung von Sequenzen, die ISR-Daten bearbeiten, diese Interrupts zu unterbinden:
  - Freigabebit speichern und löschen
  - unterbrechungsfreie Befehlsfolge
  - Ursprungswert des Freigabebits wiederherstellen.

Aufgabe 7.1: Experiment vorbereiten

- Legen Sie ein neues Projekt »TestTmrInterrupt« mit der Datei TestTmrInterrupt.c von der Web-Seite an.
- Übersetzten und im Debug-Modus starten.

Funktion des Programms:

- Die ISR wird aller Millisekunde aufgerufen und invertiert jeden 500sten Aufruf LD4 (Blinkausgabe 1 Hz).

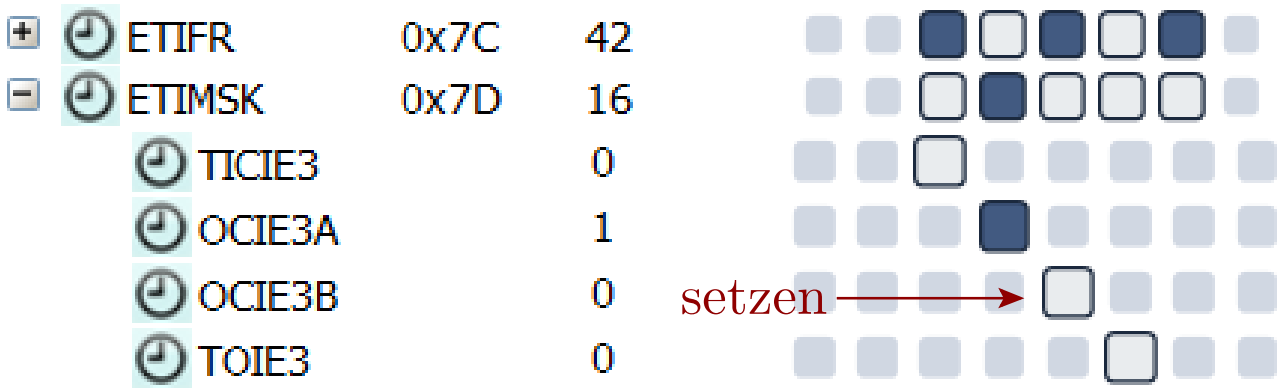
```
15  ISR(TIMER3_COMPA_vect){//ISR Timer 3 Vergleichs-
16    // interrupt B, Schrittbit OCF3A wird bei ISR-
17    // aufruf automatisch gelöscht
18    LED_Ct++;
19    if (LED_Ct>=500){ // alle 500 Schritte
20      PORTE ^= 0x10;  // LD4 invertieren
21      LED_Ct = 0;     // Zähler rücksetzen
      }                 // erzeugt 1Hz-Blinksignal
    }
```

- Das Hauptprogramm initialisiert den Ausgabeport, den Timer etc. schaltet Interrupts lokal und global ein und geht in eine Endlosschleife.

```
37  int main(void) {
38      DDRE = 0xF0;          // LD1 bis LD4 als Ausgänge
39     // Timer 3: Clear on Compare Match mit OCR3A,
40     TCCR3B = (1<<WGM32)|(0b001<<CS30); // Vorteiler 1
41     OCR3A  = 7580;        // 1 ms Aufrufperiode
42     // beim Zählen von 1 bis 7580 soll auch mit OCR3B
43     OCR3B  = 1000;        // einmal Gleichheit auftreten
44     LED_Ct = 0;
45     PORTE ^= 0x20;        // LD3 umschalten
46     ETIMSK = 1<<OCIE3A; // Timer3-Comp.B-Interrupt ein
47     sei();                // Interrupts global freigeben
48     while(1){
49  //   cli();              // Interrupts sperren
50  //   PORTE ^= 0x80;      // LD1 umschalten
51  //   sei();              // Interrupts wieder freigeben
     };
  }
```

Aufgabe 7.2: Experiment »ungewollte Interrupts«

- Stoppen Sie das Programm.
- Wählen Sie in IO-View Timer 3. und setzen Sie in ETIMSK das Bit OCIE3B: Freigabe des Interrupts, der bei Gleichheit des Zählers mit dem Register OCR3B, in dem der Wert 1000 steht, ausgelöst wird.



- Nach Fortsetzung des Programms schaltet LD3 um, d.h. main() startet neu.

  Warum? Warum nur ein Neustart?

Es gibt keine ISR für TIMER3_COMPB_vec. Der C-Compiler programmiert in diesem Fall einen Programmneustart, bei dem der zusätzliche Interrupt ausgeschaltet und LD3 invertiert wird.

Was passiert, wenn der unbehandelte Interrupt TIMER3_COMPB beim Neustart nicht ausgeschaltet wird?
Änderung Programmzeite 46 in:

```
ETIMSK |= 1<<OCIE3A;
```

Das Programm initialisiert sich immer wenn Zähler 3 den Wert 1000 erreicht (ca. alle 149µs) neu, erkennbar daran, das LD3 schwach leuchtet.

Fehlerbehandlung

Ergänzen einer ISR mit dem Vektor BADISR_vect. Der Compiler nimmt diese ISR für alle nicht explizit behandelten Interrupts.

Die nachfolgende ISR für ungewollte Interrupts invertiert LD2 bei jedem 500sten Aufrauf.

```
26  uint32_t BADISR_Ct;
27  // ISR zur Behandlung ungewollter Interrupts
28  ISR(BADISR_vect){
29    BADISR_Ct ++;
30    if (BADISR_Ct>=500){  // alle 500 Aufrufe
31      PORTE ^= 0x40;      // LD2 umschalten
32      BADISR_Ct = 0;
33    }
34  }
```

- Übersetzten und Neustart im Debugger.
- Beim Start schaltet LD3 um und LD4 fängt an zu blinken.
- Bei Setzen von OCIE3B blinkt auch LD2 mit ca. 1 Hz, d.h. ca. 1000 Aufrufe pro Sekunde.
- LD3 bleibt unverändert. Kein Neustart.

Aufgabe 7.3: Experiment Datenchaos

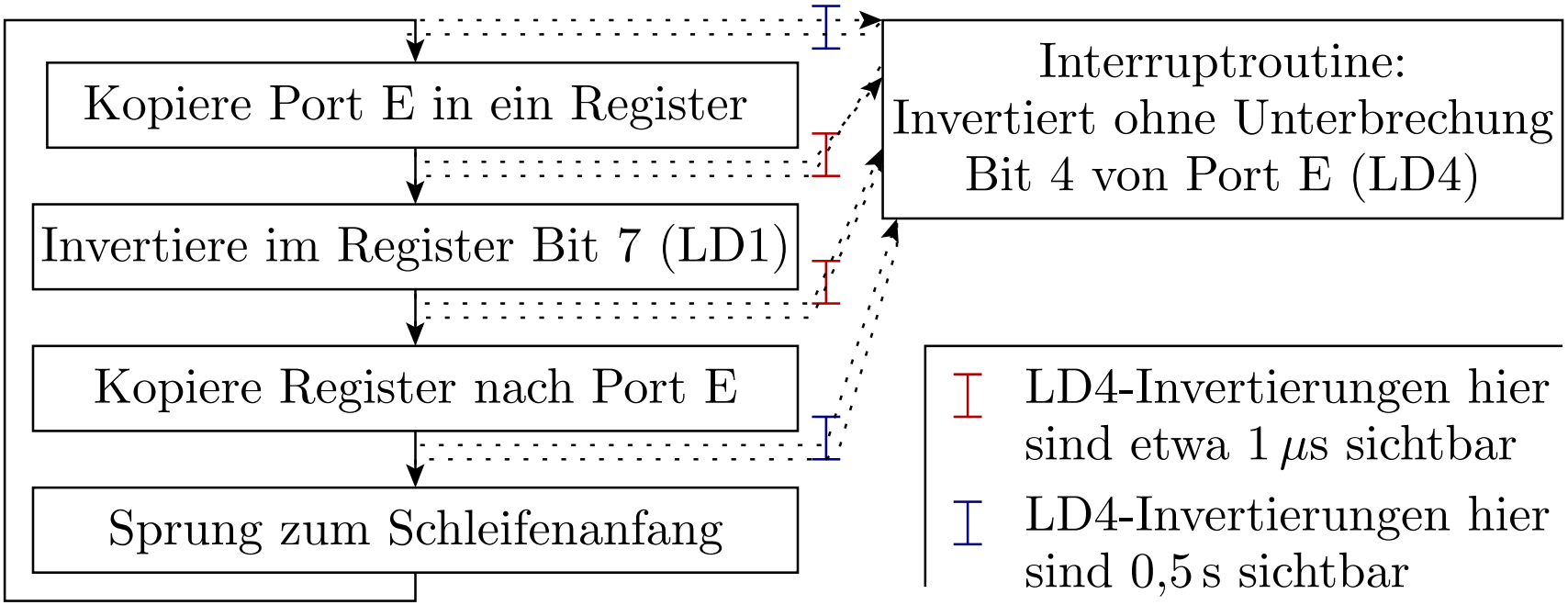
Entfernen Sie den Kommentar vor Zeile 50:

```
PORTE ^=0x80; // LD1 umschalten
```

ein und starten Sie das Programm neu. LD4 blinkt mit ca. 500kHz (leuchtet halb) und das Blinken von LD4 wird ungleichmäßig (Wechsel zwischen 1 und 2 Hz).

Warum?

Das Hauptprogramm und die ISR ändern beide den Wert von Port E. Die c-Anweisung zum invertieren von LD1 besteht aus drei Maschinenanweisungen. Plus Sprung zum Schleifenanfang umfasst der Schleifenkörper vier Anweisungen:



An 50% der Unterbrechungsmöglichkeiten wird die Invertierung von LD4 in der ISR vom Rückschreibwert für die Invertierung von LD1 überschrieben.

Unterbrechungsfreie Sequenzen

Sperren Sie Interrupts während der Invertierung von LD1:

```
48    while(1){
49      cli();              // Interrupts sperren
50      PORTE ^= 0x80;      // LD1 umschalten
51      sei();              // Interrupts wieder freigeben
52    };
```

Damit kann der Prozessor das Programm vom Lesen von Port E bis zum Schreiben des für LD1 invertierten Wertes nicht unterbrechen. Unterbrechungen vor und nach dem Sprung zum Schleifenbeginn verursachen keine Fehlfunktionen.

Nach Neuübersetzung und Neustart sind keine Blinkanormalitäten mehr beobachtbar.

LCD-Zustandsmonitor

Steuerfunktion des Fahrzeugs

Die übergeordnete Steuerfunktion des Fahrzeugs ist eine Schrittfunktion, die ganz allgemein aus

- den über Bluetooth empfangenen Steuerdaten,
- den Sensor- und Joystickeingaben und
- dem Ist-Zustand der Steuerung

etwa 10 mal in der Sekunde

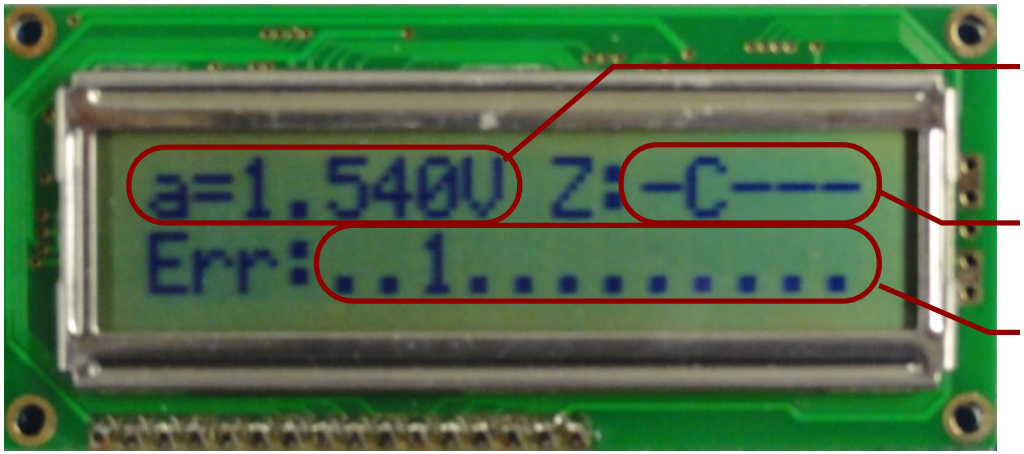
- den Folgezustand der Steuerung,
- die Antriebswerte für die Motoren und
- die über Bluetooth zu versendenden Daten

berechnet. Die erfolgreiche Inbetriebnahme erfordert eine Beobachtbarkeit der Eingaben, Zustände und Ausgaben dieser Schrittfunktion. Alternative ist eine Blindfehlersuche, bei der die nicht beobachtbaren Werte und die intern aufgetretenen Fehler zu erraten sind.

Auswahl der zu visualisierenden Daten

Das LC-Display zeigt 2×16 Zeichen an. Das erfordert eine platzsparende Darstellung und eine Beschränkung auf die wichtigsten und nicht anders beobachtbaren Daten[2]:

- Zustände der Steuerung
- Fehlerzähler
- der Wert des Abstandssensors.



[2] Anders beobachtbar sind die Daten vom und zum PC, vom Bodensensor, vom Joystick und die Motorbewegung.

Steuerzustände

Für die Steuerzustände sind fünf Zeichen reserviert, mit denen die Werte von fünf Zustandsvariablen als Ascii-Zeichen visualisiert werden. In lcd.h sind den Zustandsnummern Symbole zugeordnet:

```
37 #define Steuerzustand_A 0 // z.B. Linie gefunden,
38 #define Steuerzustand_B 1 // Hindernis rechts umfahren
39 #define Motorzustand_A  2 // für die Motorreglung: halt
40 #define Motorzustand_B  3 // vor, zurück, Schleppfehler,
41 #define Eingabezustand  4 // für den Kommandointerpreter
```

- Die Steuerzustände A und B können im eigenen Programm beliebig verwendet werden.
- Die Motorzustände werden im späteren Testbeispiel »PI-Regler« genutzt.
- Der Eingabezustand wird ab dem Projekt »Kommandointerpreter« genutzt.

Fehlerzustände

Die 12 Fehlerzähler sind mit '.' initialisiert und werden in der Reihenfolge '1', '2', ..., '9', 'a', 'b', ... bis 'z' erhöht. Beim Maximum 'z' »zu viele Fehler« bleiben die Zähler stehen.

```
25  void incErr(uint8_t errNr);
26  // 0 bis 4 unverplant, nutzbar für unplausible Steuersituationen
27  #define WM_Abstast 5  // Unterabtastung Wegemessung
28  #define UA0_DorCt  6  // Überlauf Empfangspuffer (Data OverRun)
29  #define UA0_FeCt   7  // Empfangsrahme-Fehler (Frame Error)
30  #define UA0_ParCt  8  // Empfangsparitätsfehler (PARity Error)
31  #define UA0_EByCt  9  // Empfänger besetzt
32  #define UA0_SByCt 10  // Sender besetzt
33  #define BADISR_Ct 11  // unerwünschte Interrupts
```

- Die Fehlernummern 0 bis 4 sind für das eigene Programm.
- Fehlerzähler 5 zählt im Projekt PI-Regler die Unterabtastfehler bei der Wegmessung.
- Die Fehlerzähler 6 bis 10 zählen ab dem Projekt »Kommandointerpreter« die unterschiedlichen Kommunikationsfehler.

- Unerwünschte Interrupts sind versehentlich aktivierte Interrupts, für die es keine Behandlungsroutine gibt. Sie werden in lcd.c abgefangen mit:

```
95   // Behandlung ungewollter Interrupts
96   ISR(BADISR_vect){
97     incErr(BADISR_Ct);  // Fehlerzähler erhöhen
98   }
```

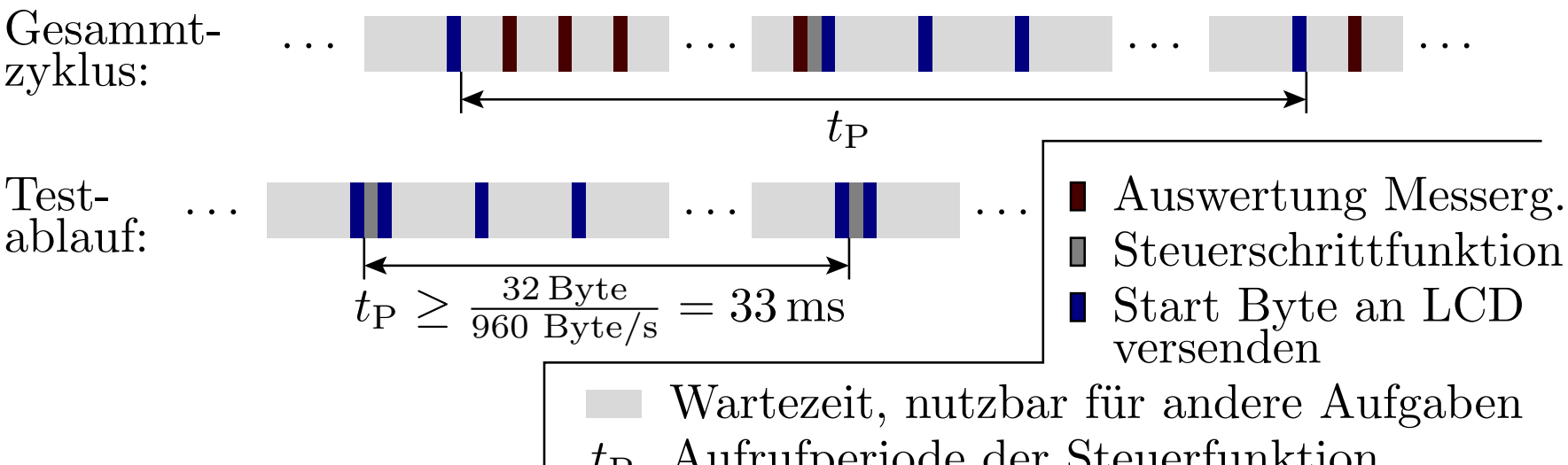
Wenn Zähler 11 anspricht, unbedingt die Ursache suchen und beseitigen, da dieser Fehler oft unkontrollierbare Nebenwirkungen auf das gesamte Programm haben. Mögliche Ursachen sind Verwechslungen von Registern, Bits und Interruptvektor, ISR vergessen zu programmieren, ...

Algorithmus und Zeitablauf

Im fertigen System wird die Zustandsausgabe Teil der Steuerschleife:

- interrupt-gesteuerte Messung der Sensorwerte.
- Abarbeitung der Steuerschrittfunktion.
- Interrupt-gesteuerte Textausgabe.

Zum Test des Monitors soll nach Versenden des letzten Bytes die Steuerfunktion, statt »Sensormessstart« aufgerufen werden.



Aufgabe 7.4: Experiment vorbereiten

- Legen Sie ein neues Projekt »TestLcdIR« an, binden Sie die Dateien für dieses Projekt von der Webseite ein.
- Schließen Sie das LC-Display an Stecker JE an. Bitte während der Umbauarbeiten an der Hardware immer Spannung ausschalten.
- Übersetzen Sie das Programm und starten Sie es im Debugger. Die auf dem LC-Display erscheinende Ausgabe sollte etwa der auf Folie 19 entsprechen.

Private Daten

```
12  // globale private Daten
13  uint8_t LCD_dat[]="a=0.000V Z:-----Err:............";
14  uint8_t *LCD_ptr;
15  // Funktionszeiger auf die Steuerfkt.
16  void (*LCD_AnschlFkt)()=0;
```

- Feld mit dem kompletten Anfangstext, in dem einzelne Werte überschrieben werden.
- Zeiger auf das Zeichen, das beim nächsten Aufruf der Puffer-frei-ISR auszugeben ist.
- Zeiger auf die Funktion, die nach Ausgabe des letzten Zeichens in der ISR aufzurufen ist.

Initialisierung USART1 und LCD

```
29   // LCD-Ausgabe initialisieren
30  void initLCD(void (*afptr)()){
31    LCD_AnschlFkt = afptr;// speichern des Funktionszeigers
32    // Initialisiere USART1 an JE mit 9600 Baud 8n1
33    UBRR1H=0; UBRR1L=49;  // 9600 Baud
34    UCSR1B = (1<<TXEN1);  // Senden ein Empfang aus
35    UCSR1C = 0b00000110;  // 8n1
37    // LCD löschen, 16 Zeichen/Zeile einstellen
38    writeLCD((uint8_t*)"\x1B[0h\x1b[j", 7);
39    writeLCD(LCD_dat, 32);
  }
```

- Übergabe Funktionszeiger Anschlussfunktion.
- USART1: 9600 Baud, 8n1, Sender ein, Interrupts aus.
- LCD:16-Spalten-Modus, Löschen, Initialwerte aus LCD_dat schreiben.

Schrittkettenfunktion

Start einer LCD-Ausgabe:

```
89  void startLCD(){
90    if (UCSR1B & (1<<UDRIE1)) return;
91    LCD_ptr = LCD_dat;       // Zeiger auf Ausgabetextanf.
92    UCSR1B |= (1<<UDRIE1);   // Puffer-frei-Interrupt ein
    }
```

Schrittfunktion für den Puffer-frei-Interrupt:

```
75  ISR(USART1_UDRE_vect){
76    if (LCD_ptr<LCD_dat+32){// solange nicht alle 32 Zeichen
77      UDR1 = *LCD_ptr;      // ausgegeben sind, Zeichenausg.
78      LCD_ptr++;            // Zeiger weiterschalten
79      PORTF ^= 1;           // Testanschluss invertieren
      }
81    else {                  // Wenn alls Zeichen ausgegeben,
82      UCSR1B &= ~(1<<UDRIE1);// Puffer-frei-Interrupt aus.
83      if (LCD_AnschlFkt)     // Wenn Anschlussfkt.-Zeig.!=0
84        (*LCD_AnschlFkt)();  // Aufruf Anschlussfunktion.
    }
  }
```

Sensor- und Zustandswerte bearbeiten

```
   // Abstandssensorwert aktualisieren
43 void updateAbstand(uint16_t w){
44   uint16_t b;
45   uint8_t idx = 2;
46   for (b=1000;b>0;b/=10) {
47     LCD_dat[idx]= w/b+'0';
48     w -= b*(w/b);
49     if (idx==2) idx+=2;
50     else idx++;
   }
}
   // Setzen eine der fünf Zustandsvariablen
55 void setState(uint8_t state, uint8_t snr){
56   if (snr<5)
57     LCD_dat[11+snr] = state;
}
60 // Lesen eine der fünf Zustandsvariablen
61 uint8_t getState(uint8_t snr){return LCD_dat[11+snr];}
```

Fehlernummer erhöhen

```
63   // Einen der 12 Fehlerzähler erhöhen
64   void incErr(uint8_t errNr){
65     if (errNr>11) return;
66     uint8_t w = LCD_dat[20+errNr];
67     if (w>='z') return;
68     else if (w>='a') w = w+1;
69     else if (w>='9') w = 'a';
70     else if (w>='1') w= w+1;
71     else w = '1';
72     LCD_dat[20+errNr] = w;
73   }
```

- Bei einer übergebenen Fehlernummer >11 Rücksprung.
- Bei jedem Aufruf das bisherige Fehlernummerzeichen aus LCD_dat gelesen und in der Reihenfolge '1', '2',..,'9','a', 'b', ... bis maximal 'z' erhöht.

Hauptprogramm, Daten der Steuerschrittfunktion

```
38  int main(void){
39    initLCD(&Schrittfunktion);
40    startLCD();         // nächste Ausgabe starten
41    sei();
42    while(1){}          // später Kommandointerpreter
    }
```

Das Hauptprogramm initialisiert und wartet. Später wird in der Hauptschleife der Kommandointerpreter laufen.

```
17  // private Daten der Schrittfunktion
18    uint32_t Abstand= 0;  // Spannung des Abstandssensors
19    uint8_t  fnr    = 2;  // Fehlernummer
20    uint8_t  znr    = 1;  // Zustandsnummer
21    uint8_t Zustand = 'D';// Zustandswert
```

Daten zum Test der Funktionen zur Aktualisierung des Abstandswertes, der angezeigten Zustände, der Fehlerzähler ...

Das Testbeispiel für die Steuerschrittfunktion

```
23  void Schrittfunktion(){
24    Abstand +=10;                 // Abstandswert hoch-
25    updateAbstand(Abstand);       // zählen und ausgeben
26    if (Abstand>3000){            // bei 3000 (3V)
27      Abstand = 0;                // rücksetzen und
28      Zustand ++;                 // Zustand weiterzählen
29      setState(Zustand, znr);     // Zustandsausgabe
30      if (Zustand>='E'){          // bei Zustand >='E'
31         Zustand = 'A';           // Folgezustand 'A' und
32       incErr(fnr);               // Fehlerzähler erhöhen
        }
      }
35    startLCD();           // nächste Ausgabe starten
    }
```

Die Spannung wird zirkular bis 3,000V gezählt, bei jedem Überlauf der Zustand weitergeschaltet und beim Wechsel von Zustand 'E' nach 'A' der Fehlerzähler erhöht.

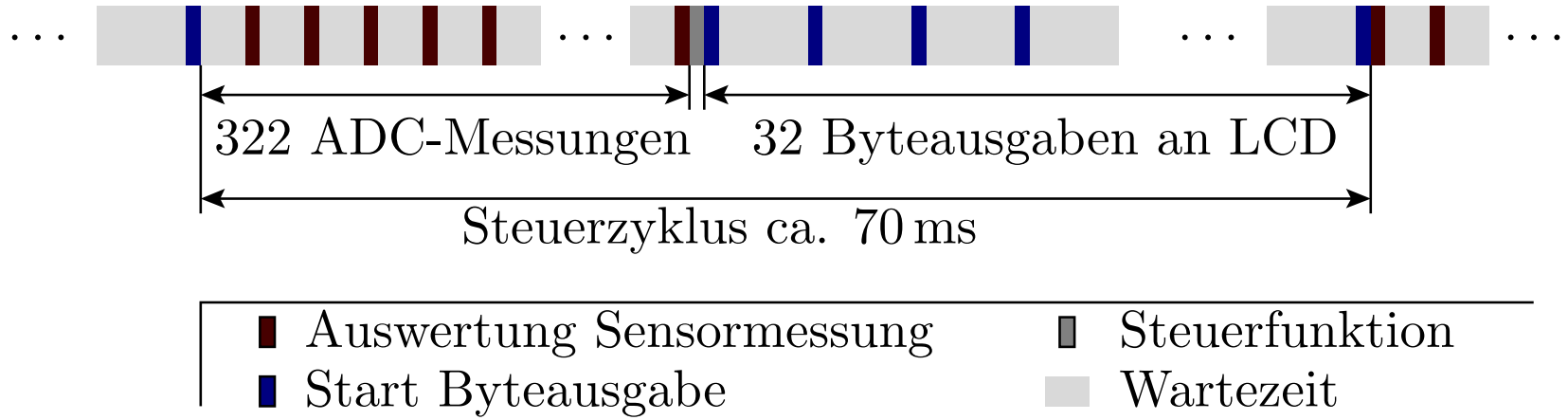
Aufgabe 7.5: Ändern der Steuerschrittfunktion

- Ändern Sie die Steuerschrittfunktion so, dass andere Zustandsvariablen und andere Fehlerzähler verändert werden.
- Untersuchen Sie, an welchen Stellen es günstig ist, Unterbrechungspunkte für die Kontrolle der Funktion zu setzen.
- Erweitern Sie das Programm so, dass sie an PF1 die Periodendauer, mit der die Steuerschrittfunktion ausgeführt wird, mit einem Multimeter messen können und messen Sie diese.

Erweiterung um die Abstandsmessung

Ablauferweiterung

Der Zyklus aus Steuerschritt und LCD-Ausgabe wird um 322-ADC-Messungen für den Abstandswert erweitert:



- Dauer der ADC-Wandlungen:

$$\frac{322\,\text{Wandlungen} \cdot 13\,\frac{\text{Schritte}}{\text{Wandlung}}}{117\,\text{kHz}} \approx 35{,}8\,\text{ms}$$

- Dauer der LCD-Ausgabe:

$$\frac{32\,\text{Byte}}{960\,\frac{\text{Byte}}{\text{s}}} \approx 33{,}3\,\text{ms}$$

Aufgabe 7.6: Experiment vorbereiten

- Legen Sie ein neues Projekt »TestLcdAbstIR« an, binden Sie die Dateien für dieses Projekt von der Webseite ein.
- Belassen Sie das LC-Display an Stecker JE und schließen Sie den Abstandssensor wie auf Foliensatz 5 an Stecker JF an (gelb Anschluss 6, schwarz Anschluss 7 und rot Anschluss 8, Jumper JPF auf VU). Bitte während der Umbauarbeiten an der Hardware immer Spannung ausschalten.
- Übersetzen Sie das Programm und starten Sie es im Debugger. Auf dem LC-Display sollte die Spannungsanzeige auf den Abstand von Hindernissen vor dem Sensor reagieren. Die auf dem LC-Display erscheinende Ausgabe sollte etwa der auf Folie 19 entsprechen.

Initialisierung der Abstandsmessung

```
13   uint16_t ADC_Ct;  // Zähler für die 322 Messungen
14   uint32_t ADC_acc; // Akkumulator Summe Messergebnisse
15   uint16_t ADC_val; // Messwert in Millivolt
16   // Funktionszeiger auf die Steuerfunktion
17   void (*ADC_SteuFkt)(uint16_t)=0;
```

Das ADC-Objekt bekommt genau wie das LCD-Objekt einen Zeiger auf eine Funktion, die beim letzten ISR-Aufruf zu starten ist

```
20 void initADC(void (*sfptr)()){
21    ADMUX = (3<<MUX0); // Kanal 3 auswählen
22    // Einschalten mit Wandlungstaktteiler 64
23    ADCSRA = (1<<ADEN)|(0b110<<ADPS0);
24    DDRE &= ~0x80;  // Sensoreingang als Eingang
25    PORTE &=~0x80;  // Ausgabewert 0 (hochohmig)
26    ADC_SteuFkt = sfptr; // Funktionszeiger speichern
27 }
```

Der ADC wird wie in Foliensatz 5 initialisiert, der bei der Initialisierung zu übergebende Funktionszeiger gespeichert.

Startfunktion:

```
30 void startADC(){
31   ADC_Ct  = 0;  // Zähler und Akkumulator löschen
32   ADC_acc = 0;  // Wandlung starten und Interrupt
33   ADCSRA |= (1<<ADIE)|(1<<ADSC); // freigeben
34 }
```

Die Startfunktion setzt den Messzähler zurück, löscht den Messwertakkumulator, schaltet den ADC ein und aktiviert die ADC-Interrupts.

Die ADC-ISR addiert das Wandlerergebnis zum Akku und erhöht den Zähler. Bei Zählstand <322 wird die nächste Wandlung gestartet, sonst der Ergebniswert aus dem akkumulierten Wert gebildet, ADC-Interrupts ausgeschaltet und die Steuerschrittfunktion aufgerufen.

Interruptfunktion

```
36 ISR(ADC_vect){
37   ADC_acc += ADC;    // Hinzusummieren des Wandlerer-
38   ADC_Ct++;          // gebnisses und Zähler erhöhen
39   if (ADC_Ct<322)    // nach weniger als 322 Wandlungen
40     ADCSRA |= (1<<ADSC); // nächste Wandlung starten
41   else {             // sonst
42     ADC_val = ADC_acc / 100;// Summe durch 100 teilen
43     ADCSRA &= ~(1<<ADIE);   // Wandlerinterrupt aus
44     updateAbstand(ADC_val); // neuen Abstandswert
45     if (*ADC_SteuFkt)       // auf LCD und wenn ein
46       (*ADC_SteuFkt)(ADC_val); // Steuerschrittfunktion
47   }                        // definiert ist, diese Aufrufen
48 }
```

Das Hauptprogramm

```
34 int main(void){
35   // Das ADC-Objekt bekommt einen Zeiger auf die Steuer-
36   initADC(&Steuerschrittfunktion); // schrittfunktion
37   // die als letztes die nächste LCD-Ausgabe startet.
38   // Das LCD-Objekt bekommt einen Zeiger auf die
39   initLCD(&startADC); // Funktion zum Start der Messungen
40   startADC(); // Start der ersten Messung
41   setState('a', Steuerzustand_A); // Steuerzustandinit.
42   sei();
43   while(1) {} // Die Hauptschleife warte nur.
44 }
```

Das Hauptprogramm legt mit der Initialisierung fest, dass der letzte ADC-ISR-Aufruf die Steuerschrittfunktion aufruft und der letzte ISR-Aufruf der LCD-Ausgabe die Startfunktion für die nächste Messung.

Die Steuerschrittfunktion

```
15  uint8_t SSF_Ct;
16  uint8_t zustand='a';

18  void Steuerschrittfunktion(uint16_t VAbst){
19    SSF_Ct ++;
20    if (SSF_Ct>=28){ // etwa alle 2 Sekunden
21      SSF_Ct = 0;
22      // Steuerzustand_A zirkular erhöhen: a,b,..,z,a,..
23      zustand = getState(Steuerzustand_A);
24      if (zustand<='z') zustand++;
25      else zustand = 'a';
26      setState(zustand, Steuerzustand_A);
27      if (VAbst>2000) // Wenn Abstandsspannung größer 2V
28        incErr(2);    // Fehlerzähler 2 erhöhen
    }
30  startLCD();
}
```

Als letztes wird die nächste LCD-Ausgabe gestartet.

Aufgabe 7.7: Steuerschrittfunktion

- Beschreiben Sie, wie im Beispiel die Steuerschrittfunktion aus Eingaben und Ist-Zuständen Ausgaben und Folgezustände berechnet.
- Erweitern Sie die Steuerschrittfunktion so, dass auch auf die Bodensensoren eine sichtbare Reaktion erfolgt.