Übungen zu Systemnahe Programmierung in C

Abschnitt 7.2: Synchronisation

08.06.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Schlüsselwort volatile

- Bei einem Interrupt wird `event = 1` gesetzt
- Aktive Warteschleife wartet, bis `event != 0`
- Der Compiler erkennt, dass `event` innerhalb der Warteschleife nicht verändert wird
  - ⇒ der Wert von event wird nur einmal vor der Warteschleife aus dem Speicher in ein Prozessorregister geladen
  - ⇒ Endlosschleife

```
01 static uint8_t event = 0;
02 ISR(INT0_vect) {
03     event = 1;
04 }
05
06 void main(void) {
07     while(1) {
08         while(event == 0) { /* warte auf Event */ }
09         // bearbeite Event [...]
10     }
11 }
```

Schlüsselwort volatile

- Bei einem Interrupt wird `event = 1` gesetzt
- Aktive Warteschleife wartet, bis `event != 0`
- Der Compiler erkennt, dass `event` innerhalb der Warteschleife nicht verändert wird
  - ⇒ der Wert von event wird nur einmal vor der Warteschleife aus dem Speicher in ein Prozessorregister geladen
  - ⇒ Endlosschleife
- `volatile` erzwingt das Laden bei jedem Lesezugriff

```
01 static volatile uint8_t event = 0;
02 ISR(INT0_vect) {
03     event = 1;
04 }
05
06 void main(void) {
07     while(1) {
08         while(event == 0) { /* warte auf Event */ }
09         // bearbeite Event [...]
10     }
11 }
```

Verwendung von volatile

- Fehlendes `volatile` kann zu unerwartetem Programmablauf führen
- Unnötige Verwendung von `volatile` unterbindet Optimierungen des Compilers
- Korrekte Verwendung von `volatile` ist Aufgabe des Programmierers!

⇝ Verwendung von `volatile` so selten wie möglich, aber so oft wie nötig

→ 2.1

Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
01 static volatile uint8_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07   while(1) {
08     if(counter > 0) {
09
10       counter--;
11
12       // verarbeite Tastendruck
13       // [...]
14     }
15   }
16 }
```

Lost Update

Hauptprogramm

```
01  ; C-Anweisung: counter--;
02  lds r24, counter
03  dec r24
04  sts counter, r24
```

Interruptbehandlung

```
05  ; C-Anweisung: counter++
06  lds r25, counter
07  inc r25
08  sts counter, r25
```

INT →

Zeile	counter	r24	r25
—	5	—	—
2	5	5	
3	5	4	
6	5	4	5
7	5	4	6
8	6	4	6
4	4	4	6

Lost Update: counter ≠ 5

16-Bit Zugriffe (Read-Write)

(8 Bit AVR)

- Nebenläufige Nutzung von 16-Bit Werten (Read-Write)
  - Inkrementierung in der Unterbrechungsbehandlung
  - Auslesen im Hauptprogramm

```
01 static volatile uint16_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07   if(counter > 300) {
08     sb_led_on(YELLOW0);
09   } else {
10     sb_led_off(YELLOW0);
11   }
12
13   // [...]
14 }
```

16-Bit Zugriffe (Read-Write)

Hauptprogramm

```
01  ; C-Anweisung: if(counter > 300)
02  lds  r22, counter
03  lds  r23, counter+1
04  cpi  r22, 0x2D
05  sbci r23, 0x01
```

Interruptbehandlung

```
07  ; C-Anweisung: counter++;
08  lds r24, counter
09  lds r25, counter+1
10  adiw r24,1
11  sts counter+1, r25
12  sts counter, r24
```

INT →

Zeile	counter	r22 & r23	r24 & r25
–	0x00ff	–	–
2	0x00ff	0x-ff	
8 & 9	0x00ff	0x-ff	0x00ff
10	0x00ff	0x-ff	0x0100
11 & 12	0x0100	0x-ff	0x0100
3	0x0100	0x01ff	0x0100

0x012D

Read Write Anomalie: counter > 300

Sperren der Unterbrechungsbehandlung beim AVR

- Viele weitere Nebenläufigkeitsprobleme möglich
  - nicht-atomare Modifikation von gemeinsamen Daten
  - Problemanalyse durch den Anwendungsprogrammierer
  - Auswahl geeigneter Synchronisationsprimitive
- Lösung hier: Einseitiger Ausschluss durch Sperren der Interrupts
  - Sperrung aller Interrupts: `cli()` und `sei()`
  - Maskieren einzelner Interrupts (`EIMSK`-Register)
- Problem: Interrupts während der Sperrung gehen evtl. verloren → 7.1

⇒ Kritische Abschnitte müssen so kurz wie möglich sein

Lost Update

- Wie kann man das Lost Update verhindern?

```
01 static volatile uint8_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07   while(1) {
08     if(counter > 0) {
09
10       counter--;
11
12       // verarbeite Tastendruck
13       // [...]
14     }
15   }
16 }
```

← nicht atomar! => Kritischer Abschnitt

Lost Update

- Wie kann man das Lost Update verhindern?

```
01 static volatile uint8_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07   while(1) {
08     if(counter > 0) {
09       cli();
10       counter--;
11       sei();
12       // verarbeite Tastendruck
13       // [...]
14     }
15   }
16 }
```

← nicht atomar! => Kritischer Abschnitt

16-Bit Zugriffe (Read-Write)

- Wie kann man die Read-Write Anomalie verhindern?

```
01 static volatile uint16_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07
08
09
10   if(counter > 300) {
11
12     sb_led_on(YELLOW0);
13   } else {
14
15     sb_led_off(YELLOW0);
16   }
17
18   // [...]
19 }
```

← nicht atomar! => Kritischer Abschnitt

16-Bit Zugriffe (Read-Write)

- Wie kann man die Read-Write Anomalie verhindern?

```
01 static volatile uint16_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07   cli();
08   uint16_t local_counter = counter;
09   sei();
10   if(local_counter > 300) {
11
12     sb_led_on(YELLOW0);
13   } else {
14
15     sb_led_off(YELLOW0);
16   }
17
18   // [...]
19 }
```

16-Bit Zugriffe (Read-Write)

- Wie kann man die Read-Write Anomalie verhindern?

```
01 static volatile uint16_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07
08
09   cli();
10   if(counter > 300) {
11
12     sb_led_on(YELLOW0);
13   } else {
14
15     sb_led_off(YELLOW0);
16   }
17   sei();
18   // [...]
19 }
```

16-Bit Zugriffe (Read-Write)

- Wie kann man die Read-Write Anomalie verhindern?

```
01 static volatile uint16_t counter = 0;
02 ISR(INT0_vect) {
03   counter++;
04 }
05
06 void main(void) {
07
08
09   cli();
10   if(counter > 300) {
11     sei();
12     sb_led_on(YELLOW0);
13   } else {
14     sei();
15     sb_led_off(YELLOW0);
16   }
17
18   // [...]
19 }
```