Übungen zu Systemnahe Programmierung in C (SPiC)

Moritz Strübe, Rainer Müller
(Lehrstuhl Informatik 4)

Sommersemester 2013

Inhalt

Interrupts
- Allgemein
- AVR
- Interrupt-Handler

Synchronisation
- volatile
- Lost Update
- 16-Bit-Zugriffe
- Sperren von Interrupts

Stromsparmodi

Interrupts

- Ablauf eines Interrupts (Vgl. 15-7)

0. Hardware setzt entsprechendes Flag
1. Sind die Interrupts aktiviert und der Interrupt nicht maskiert unterbricht der Interruptcontroller die aktuelle Ausführung
2. weitere Interrupts werden deaktiviert
3. aktuelle Position im Programm wird gesichert
4. Eintrag im Interrupt-Vektor ermitteln
5. Die Adresse im Interrupt-Vektor wird angesprungen (= Sprung in den Interrupt-Handler)
6. am Ende der Bearbeitungsfunktion bewirkt ein Befehl "Return from Interrupt" die Fortsetzung des Anwendungsprogramms und die Reaktivierung der Interrupts

Implementierung von Interruptbehandlungen

- Je Interrupt steht ein Bit zum Zwischenspeichern zur Verfügung
- Ursachen für den Verlust von Interrupts
  - Während einer Interruptbehandlung
  - Interruptsperren (zur Synchronisation von kritischem Abschnitten)
- Das Problem ist generell nicht zu verhindern

  ⇝ Risikominimierung: Interruptbehandlungen sollten möglichst kurz sein
  - Schleifen und Funktionsaufrufe vermeiden
  - Auf blockierende Funktionen Verzicheten (ADC/serielle Schnittstelle!)
- `sei` sollte niemals in einer Interruptbehandlung ausgeführt werden
  - potentiell endlos geschachtelte Interruptbehandlung
  - Stackoverflow möglich (Vorlesung,vorraussichtlich Kapitel 17)

Interrupts beim AVR

- Timer
- Serielle Schnittstelle
- ADC (Analog-Digital-Umsetzer)
- Externe Interrupts durch Pegel(änderung) an bestimmten I/O-Pins

  ⇒ ATmega32: 3 Quellen an den Pins `PD2`, `PD3` und `PB2`
  - Pegel- oder flankengesteuert
  - Abhängig von der jeweiligen Interruptquelle
  - Konfiguration über Bits
  - Beispiel: Externer Interrupt 2 (`INT2`)

`ISC2`	IRQ bei:
0	fallender Flanke
1	steigender Flanke

- Dokumentation im ATmega32-Datenblatt
  - Interruptbehandlung allgemein: S. 45-49
  - Externe Interrupts: S. 69-72

(De-) Aktivieren von Interrupts beim AVR

- Interrupts können durch die spezielle Maschinenbefehle aktiviert bzw. deaktiviert werden.
- Die Bibliothek avr-libc bietet hierfür Makros an:

```
#include <avr/interrupt.h>
```

  - `sei()` (Set Interrupt Flag) - lässt ab dem nächsten Takt Interrupts zu
  - `cli()` (Clear Interrupt Flag) - blockiert (sofort) alle Interrupts
- Beim Betreten eines Interrupt-Handlers werden automatisch alle Interrupts blockiert, beim Verlassen werden sie wieder deblockiert
- Beim Start des µC sind die Interrupts abgeschaltet

INT2 konfigurieren

- Beim ATmega32 verteilen sich die Interrupt Sense Control (`ISC`)-Bits zur Konfiguration der externen Interrupts auf zwei Register:
  - MCU Control Register (`MCUCR`)
  - MCU Control and Status Register (`MCUCSR`)
- Position der ISC-Bits in den Registern durch Makros definiert `ISCn0` und `ISCn1` (`INT0` und `INT1`) oder `ISC2` (`INT2`)
- Beispiel: `INT2` bei ATmega32 für fallende Flanke konfigurieren

```
1 /* die ISCs für INT2 befinden sich im MCUCSR */
2 MCUCSR &= ~(1<<ISC2); /* ISC2 löschen */
```

(De-) Maskieren von Interrupts

- Einzelne Interrupts können separat aktiviert (=demaskiert) werden
  - ATmega32: General Interrupt Control Register (`GICR`)
- Die Bitpositionen in diesem Register sind durch Makros `INTn` definiert
- Ein gesetztes Bit aktiviert den jeweiligen Interrupt
- Beispiel: Interrupt 2 aktivieren

```
1 GICR |= (1<<INT2); /* demaskiere Interrupt 2 */
```

Interrupt-Handler

- Installieren eines Interrupt-Handlers wird durch C-Bibliothek unterstützt
- Makro `ISR` (Interrupt Service Routine) zur Definition einer Handler-Funktion (`#include <avr/interrupt.h>`)
- Parameter: gewünschten Vektor; z. B. `INT2_vect` für externen Interrupt 2
  - verfügbare Vektoren: siehe avr-libc-Doku zu `avr/interrupt.h`
  - verlinkt im Doku-Bereich auf der SPiC-Webseite
- Beispiel: Handler für Interrupt 2 implementieren

```
1 #include <avr/interrupt.h>
2 static uint16_t zaehler = 0;
3
4 ISR (INT2_vect){
5         zaehler++;
6 }
```

Das volatile-Schlüsselwort

- 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

```
1  static uint8_t event = 0;
2  ISR (INT0_vect) { event = 1; }
3
4  void main(void) {
5      while(1) {
6          while(event == 0) { /* warte auf Event */ }
7          /* bearbeite Event */
```

Das volatile-Schlüsselwort

- 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

```
1  volatile static uint8_t event = 0;
2  ISR (INT0_vect) { event = 1; }
3
4  void main(void) {
5      while(1) {
6          while(event == 0) { /* warte auf Event */ }
7          /* bearbeite Event */
```

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

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		
4	5	5	-

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		
4	5	5	-
5	5	4	-

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		
4	5	5	-
5	5	4	-
10	5	4	5

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		
4	5	5	-
5	5	4	-
10	5	4	5
11	5	4	6

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		
4	5	5	-
5	5	4	-
10	5	4	5
11	5	4	6
12	6	4	6

Lost Update

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

```
1  ; Hauptprogramm
2  volatile uint8_t zaehler;
3  ; C-Anweisung: zaehler--;
4  lds r24, zaehler
5  dec r24
6  sts zaehler, r24
```

```
7  ; Interrupt-Behandlung
8
9  ; C-Anweisung: zaehler++
10 lds r25, zaehler
11 inc r25
12 sts zaehler, r25
```

Zeile	zaehler	zaehler HP	zaehler INT
-	5		
4	5	5	-
5	5	4	-
10	5	4	5
11	5	4	6
12	6	4	6
6	4	4	-

16-Bit-Zugriffe

- Nebenläufige Nutzung von 16-Bit-Werten

```
1  ; Hauptprogramm
2  volatile uint16_t zaehler;
3
4  ; C-Anweisung: z=zaehler;
5  lds r22, zaehler
6  lds r23, zaehler+1
7  ; Verwendung von z
```

```
8  ; Interrupt-Behandlung
9  ; C-Anweisung: zaehler++
10 lds r24, zaehler
11 lds r25, zaehler+1
12 adiw r24,1
13 sts zaehler+1, r25
14 sts zaehler, r24
```

Zeile	zaehler	z HP
-	0x00ff	

16-Bit-Zugriffe

- Nebenläufige Nutzung von 16-Bit-Werten

```
1  ; Hauptprogramm
2  volatile uint16_t zaehler;
3
4  ; C-Anweisung: z=zaehler;
5  lds r22, zaehler
6  lds r23, zaehler+1
7  ; Verwendung von z
```

```
8  ; Interrupt-Behandlung
9  ; C-Anweisung: zaehler++
10 lds r24, zaehler
11 lds r25, zaehler+1
12 adiw r24,1
13 sts zaehler+1, r25
14 sts zaehler, r24
```

Zeile	zaehler	z HP
-	0x00ff	
4	0x00ff	0x??ff

16-Bit-Zugriffe

- Nebenläufige Nutzung von 16-Bit-Werten

```
1  ; Hauptprogramm
2  volatile uint16_t zaehler;
3
4  ; C-Anweisung: z=zaehler;
5  lds r22, zaehler
6  lds r23, zaehler+1
7  ; Verwendung von z
```

```
8  ; Interrupt-Behandlung
9  ; C-Anweisung: zaehler++
10 lds r24, zaehler
11 lds r25, zaehler+1
12 adiw r24,1
13 sts zaehler+1, r25
14 sts zaehler, r24
```

Zeile	zaehler	z HP
-	0x00ff	
4	0x00ff	0x??ff
10 - 14	0x0100	0x??ff

16-Bit-Zugriffe

- Nebenläufige Nutzung von 16-Bit-Werten

```
1  ; Hauptprogramm
2  volatile uint16_t zaehler;
3
4  ; C-Anweisung: z=zaehler;
5  lds r22, zaehler
6  lds r23, zaehler+1
7  ; Verwendung von z
```

```
8  ; Interrupt-Behandlung
9  ; C-Anweisung: zaehler++
10 lds r24, zaehler
11 lds r25, zaehler+1
12 adiw r24,1
13 sts zaehler+1, r25
14 sts zaehler, r24
```

Zeile	zaehler	z HP
-	0x00ff	
4	0x00ff	0x??ff
10 - 14	0x0100	0x??ff
6	0x0100	0x01ff

$\Rightarrow$ Abweichung um 255 !

Sperren der Unterbrechungsbehandlung beim AVR

- Viele weitere Nebenläufigkeitsprobleme möglich
  - Nicht-atomare Modifikation von gemeinsamen Daten kann zu Inkonsistenzen führen
  - Problemanalyse durch den Anwendungsprogrammierer
  - Auswahl geeigneter Synchronisationsprimitive
- Lösung hier: Einseitiger Ausschluss durch Sperren der Interrupts
  - Sperrung aller Interrupts (`sei()`, `cli()`)
  - Maskieren einzelner Interrupts (`GICR`-Register)
- Problem: Interrupts während der Sperrung gehen evtl. verloren
  - Kritische Abschnitte sollten so kurz wie möglich gehalten werden.

Stromsparmodi von AVR-Prozessoren

- AVR-basierte Geräte oft batteriebetrieben (z.B. Fernbedienung)
- Energiesparen kann die Lebensdauer drastisch erhöhen
- AVR-Prozessoren unterstützen unterschiedliche Powersave-Modi
  - Deaktivierung funktionaler Einheiten
  - Unterschiede in der "Tiefe"des Schlafes
  - Nur aktive funktionale Einheiten können die CPU aufwecken
- Standard-Modus: Idle
  - CPU-Takt wird angehalten
  - Keine Zugriffe auf den Speicher
  - Hardware (Timer, externe Interrupts, ADC, etc) sind weiter aktiv
- Dokumentation im ATmega32-Datenblatt, S. 33-37

Nutzung der Sleep-Modi

- Unterstützung aus der avr-libc: (`#include <avr/sleep.h>`)
  - `sleep_enable()` - aktiviert den Sleep-Modus
  - `sleep_cpu()` - setzt das Gerät in den Sleep-Modus
  - `sleep_disable()` - deaktiviert den Sleep-Modus
  - `set_sleep_mode(uint8_t mode)` - stellt den zu verwendenden Modus ein
- Dokumentation von `avr/sleep.h` in avr-libc-Dokumentation
  - verlinkt im Doku-Bereich auf der SPiC-Webseite
- Beispiel

```
1 #include <avr/sleep.h>
2 set_sleep_mode(SLEEP_MODE_IDLE); /* Idle-Modus verwenden */
3 sleep_enable();                  /* Sleep-Modus aktivieren */
4 sleep_cpu();                     /* Sleep-Modus betreten */
5 sleep_disable();                 /* "Empfohlen": Sleep-Modus danach
      deaktivieren */
```

- Lost wakeup $\rightsquigarrow$ Vorlesung [15-20ff]