

Übungen zu Systemnahe Programmierung in C

Abschnitt 7.1: Interrupts

08.06.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



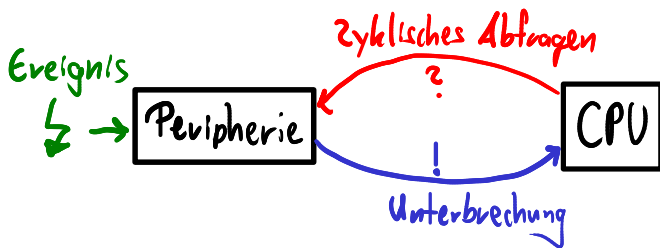
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Vorsicht, die Annotation in diesem PDF können sich vom Video unterscheiden.
Das Original ist leider verloren gegangen...

Interrupts

vs. Polling



- Ablauf eines Interrupts:  VL18
 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. Adresse des Handlers wird aus Interrupt-Vektor-Tabelle gelesen und angesprungen
 5. Ausführung des Interrupt-Handlers
 6. Am Ende des Handlers bewirkt ein Befehl "Return from Interrupt" die Fortsetzung des Anwendungsprogramms und die Reaktivierung der Interrupts



- Je Interrupt steht ein Bit zum Zwischenspeichern zur Verfügung
 - Ursachen für den Verlust von weiteren Interrupts
 - Während Interrupt-Handlers ausgeführt wird
 - Interruptsperrern (zur Synchronisation von kritischen Abschnitten) ~7.2
 - Das Problem ist generell nicht zu verhindern
- ~ Risikominimierung: Interrupt-Handler sollten möglichst kurz sein
- Schleifen und Funktionsaufrufe vermeiden
 - Auf blockierende Funktionen verzichten (ADC/serielle Schnittstelle!)

Interrupts beim AVR



- Timer
- Serielle Schnittstelle
- ADC (Analog-Digital-Umsetzer)
- Externe Interrupts durch Pegel (-änderung) an bestimmten I/O-Pins
 - Wahlweise pegel- oder flankengesteuert
 - Abhängig von der jeweiligen Interruptquelle
 - ⇒ ATmega328PB: 2 Quellen an den Pins PD2 (INT0) und PD3 (INT1)
 - ⇒ BUTTON0 an PD2
 - ⇒ BUTTON1 an PD3
- Dokumentation im ATmega328PB-Datenblatt



- 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 Interrupts zu (Um eine Instruktion verzögert)
 - `cli()` (Clear Interrupt Flag): blockiert alle Interrupts (sofort)
- Beim Betreten eines Interrupt-Handlers werden automatisch alle Interrupts blockiert, beim Verlassen werden sie wieder freigeschalten
- `sei()` sollte niemals in einer Interruptbehandlung ausgeführt werden
 - Potentiell endlos geschachtelte Interruptbehandlung
 - Stackoverflow möglich
- Beim Start des μC sind die Interrupts abgeschaltet

→ 7.3

- Interrupt Sense Control (ISC) Bits befinden sich beim ATmega328PB im External Interrupt Control Register A (EICRA)
- Position der ISC-Bits im Register durch Makros definiert

Interrupt <u>INT0</u>		Interrupt bei	Interrupt <u>INT1</u>	
<u>ISC01</u>	<u>ISC00</u>		ISC11	ISC10
0	0	low Pegel	0	0
0	1	beliebiger Flanke	0	1
1	0	<u>fallender Flanke</u>	<u>1</u>	<u>0</u>
1	1	steigender Flanke	1	1

- Beispiel: INT1 bei ATmega328PB für fallende Flanke konfigurieren

```
01 /* die ISC-Bits befinden sich im EICRA */
02 EICRA &= ~(1 << ISC10); // ISC10 löschen
03 EICRA |= (1 << ISC11); // ISC11 setzen
```


- Einzelne Interrupts können separat aktiviert (=demaskiert) werden
 - ATmega328PB: External Interrupt Mask Register (EIMSK)
- Die Bitpositionen in diesem Register sind durch Makros `INTn` definiert
- Ein gesetztes Bit aktiviert den jeweiligen Interrupt
- Beispiel: Externen Interrupt INT1 aktivieren

```
01 EIMSK |= (1 << INT1); // Demaskiere externen Interrupt INT0
```

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: Gewünschter Vektor
 - Verfügbare Vektoren: Siehe avr-libc-Doku zu `avr/interrupt.h`
 - Beispiel: INT1_vect für externen Interrupt INT1
- Beispiel: Handler für INT1 implementieren

```
01 #include <avr/interrupt.h>
02
03 static volatile uint16_t zaehler = 0;
04
05 ISR(INT1_vect) {
06     zaehler++;
07 }
```

→ 7.2