

# Interrupts

„Halt! Stop! Jetzt rede ich!“

# Gliederung

- Was sind Interrupts
- Wodurch werden sie ausgelöst?
- Wofür kann ich sie verwenden?
- Wie muss ich sie einstellen?
- Was sollte ich beachten
- Quellen

## Was sind Interrupts?

- Ist eine Unterbrechungsaufforderung
- Tritt ein bestimmtes Ereignis ein wird er ausgelöst
- Das Programm wird unterbrochen und danach fortgesetzt

Wodurch werden sie ausgelöst?

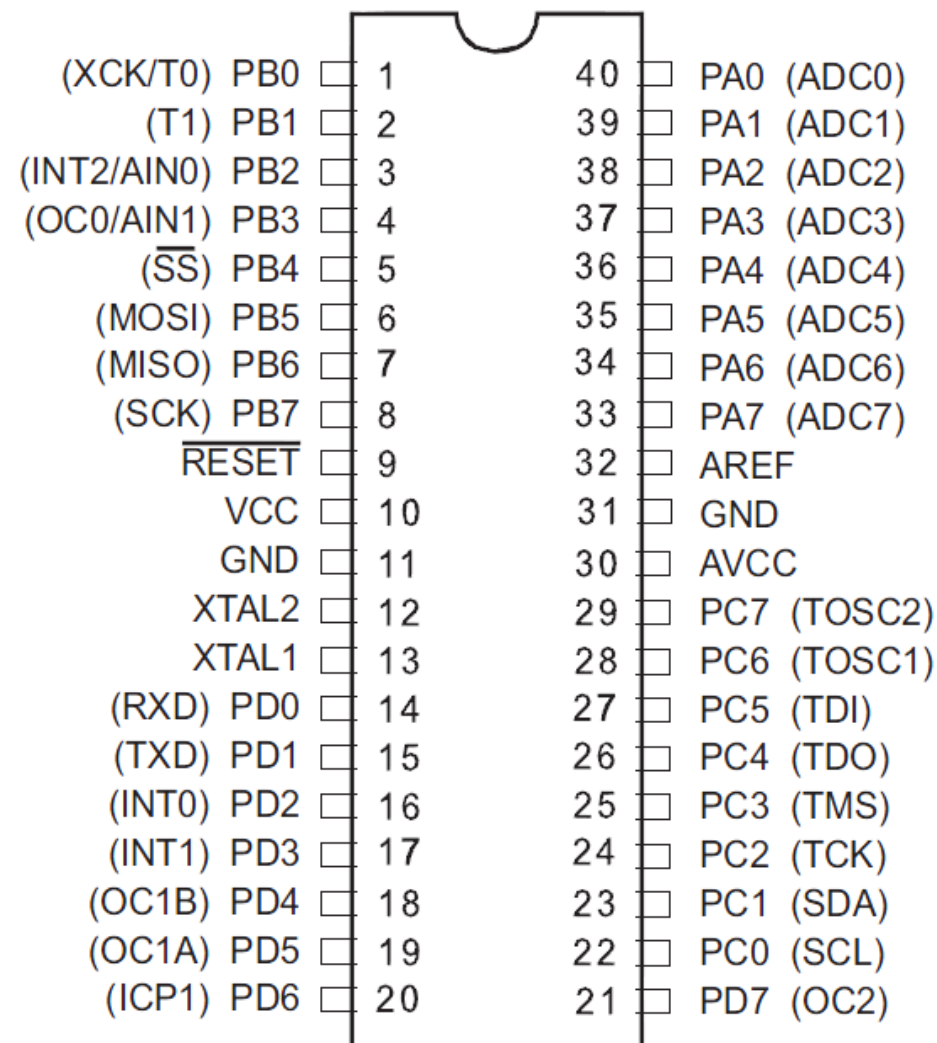
- An einem bestimmten Eingang ändert sich ein Zustand
- Eine Übertragung ist beendet
- Eine gewählte Zeitspanne wurde überschritten
- etc.

Wie stelle ich Interrupts ein?

Für INT0 und INT1:

- Maskenbit setzen
- Interruptbearbeitung definieren
- Interrupts global freigeben

Danach muss der Interrupt natürlich noch eintreffen.



# 1. Maskenbits setzen

- Benötigt werden GICR und MCUCR

## General Interrupt Control Register – GICR

| Bit           | 7    | 6    | 5    | 4 | 3 | 2 | 1     | 0    |      |
|---------------|------|------|------|---|---|---|-------|------|------|
|               | INT1 | INT0 | INT2 | – | – | – | IVSEL | IVCE | GICR |
| Read/Write    | R/W  | R/W  | R/W  | R | R | R | R/W   | R/W  |      |
| Initial Value | 0    | 0    | 0    | 0 | 0 | 0 | 0     | 0    |      |

## MCU Control Register – MCUCR

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

| Bit           | 7   | 6   | 5   | 4   | 3     | 2     | 1     | 0     |       |
|---------------|-----|-----|-----|-----|-------|-------|-------|-------|-------|
|               | SM2 | SE  | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write    | R/W | R/W | R/W | R/W | R/W   | R/W   | R/W   | R/W   |       |
| Initial Value | 0   | 0   | 0   | 0   | 0     | 0     | 0     | 0     |       |

- IVSEL: Interrupt Vector Select
  - 0: Anfang des Flash
  - 1: Anfang der Flash Boot Loader Section  
(gesteuert von den BOOTSZ Fuses)
- IVCE: Interrupt Vector Chance Enable
  - Möchte IVSEL gesetzt werden so muss IVCE 1 sein



- Codebeispiel Atmega16 Datenblatt

#### C Code Example

```
void Move_interrupts(void)
{
    /* Enable change of interrupt vectors */
    GICR = (1<<IVCE);
    /* Move interrupts to boot Flash section */
    GICR = (1<<IVSEL);
}
```

## Auszug Interrupt Vektor Tabelle

| Vector No. | Program Address <sup>(2)</sup> | Source       | Interrupt Definition  |
|------------|--------------------------------|--------------|---|
| 1          | \$000 <sup>(1)</sup>           | RESET        | External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset |
| 2          | \$002                          | INT0         | External Interrupt Request 0  |
| 3          | \$004                          | INT1         | External Interrupt Request 1  |
| 4          | \$006                          | TIMER2 COMP  | Timer/Counter2 Compare Match  |
| 5          | \$008                          | TIMER2 OVF   | Timer/Counter2 Overflow   |
| 6          | \$00A                          | TIMER1 CAPT  | Timer/Counter1 Capture Event  |
| 7          | \$00C                          | TIMER1 COMPA | Timer/Counter1 Compare Match A  |
| 8          | \$00E                          | TIMER1 COMPB | Timer/Counter1 Compare Match B  |
| 9          | \$010                          | TIMER1 OVF   | Timer/Counter1 Overflow   |

- Einstellungsmöglichkeiten MCUCR

| ISC11 | ISC10 | Description  |
|-------|-------|--|
| 0     | 0     | The low level of INT1 generates an interrupt request.      |
| 0     | 1     | Any logical change on INT1 generates an interrupt request. |
| 1     | 0     | The falling edge of INT1 generates an interrupt request.   |
| 1     | 1     | The rising edge of INT1 generates an interrupt request.    |

| ISC01 | ISC00 | Description  |
|-------|-------|--|
| 0     | 0     | The low level of INT0 generates an interrupt request.      |
| 0     | 1     | Any logical change on INT0 generates an interrupt request. |
| 1     | 0     | The falling edge of INT0 generates an interrupt request.   |
| 1     | 1     | The rising edge of INT0 generates an interrupt request.    |

## 2. Interrupt Abarbeitung definieren

- Wie oben gesehen mit Interrupt Vektoren
- Diese sind fest definiert
- Heißen im engl. **Interrupt Service Routine**

## In Assembler:

```
.include „m16def.inc“ #Prozessortyp definieren
```

```
.org 0x000
```

```
    rjmp main    #resethandler
```

```
    rjmp int0_handler
```

```
    reti
```

```
main:    rjmp main
```

```
int0_handler: cbi PORTB, 0
```

```
    reti
```

## Wichtig!

- Es darf KEINE Interruptroutine ausgelassen werden
- Wird eine vergessen zu definieren, so „verrutschen“ die Interrupt fälle
- Bei keiner Benutzung „reti“ einfügen

In C:

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

```
ISR(INT0_vect){
```

```
//Code für den Interrupt
```

```
}
```

### 3. Interrupts global einschalten

#### In Assembler:

- sei (für on)
- cli (für off)

#### In C:

- sei();
- cli();
- Bsp:     sei();.....  
             cli();  
             einlesen();  
             sei();



Was muss ich beachten

- Interruptroutinen immer so kurz wie möglich halten
- Dürfen im max. so lange dauern wie die kürzeste Dauer des Ereignisses
- In die Interruptzeit immer etwas Puffer einrechnen
- Dauer kann z.b. mit einem Oszilloskop gemessen oder simuliert werden

## Quellen

- Datenblatt Atmega16
- [http://www.mikrocontroller.net/articles/AVR-Tutorial:\\_Interrupts](http://www.mikrocontroller.net/articles/AVR-Tutorial:_Interrupts) (12.05.12, 12:42)
- <http://www.mikrocontroller.net/articles/Interrupt> (12.05.12, 12:31)
- <http://de.wikipedia.org/wiki/Interrupt> (12.05.12, 12:43)

Vielen Dank für die Aufmerksamkeit

Fragen?

Kommentare?

Anregungen?