

# Systemnahe Programmierung in C (SPiC)

## 19 Unterbrechungen – Beispiel

**Jürgen Kleinöder, Daniel Lohmann, Volkmar Sieh**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität  
Erlangen-Nürnberg

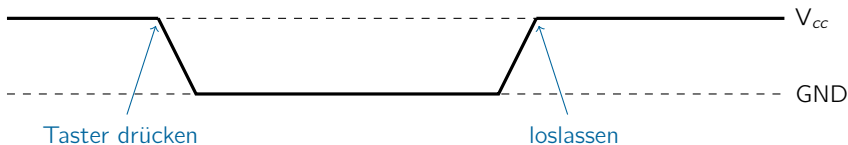
Sommersemester 2020

[http://www4.cs.fau.de/Lehre/SS20/V\\_SPiC](http://www4.cs.fau.de/Lehre/SS20/V_SPiC)



# Pegel- und Flanken-gesteuerte Interrupts

- Beispiel: Signal eines **idealisierten** Tasters (*active low*)



- Flankengesteuerter Interrupt
  - Interrupt wird durch den Pegelwechsel (Flanke) ausgelöst
  - Häufig ist konfigurierbar, welche Flanke (steigend/fallend/beide) einen Interrupt auslösen soll
- Pegelgesteuerter Interrupt
  - Interrupt wird immer wieder ausgelöst, so lange der Pegel anliegt



# Interruptsteuerung beim AVR ATmega

## ■ IRQ-Quellen beim ATmega328PB

- 45 IRQ-Quellen
- einzeln de-/aktivierbar
- IRQ  $\rightsquigarrow$  Sprung an Vektor-Adresse

(IRQ  $\rightsquigarrow$  Interrupt ReQuest)

[1, S. 78]

## ■ Verschaltung SPiCboard

(  $\leftrightarrow$  17-12  $\leftrightarrow$  2-9 )

- INT0  $\rightsquigarrow$  PD2  $\rightsquigarrow$  Button0 (hardwareseitig entprellt)
- INT1  $\rightsquigarrow$  PD3  $\rightsquigarrow$  Button1

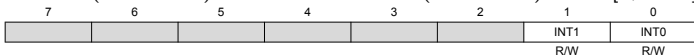
Vector No	Program Address	Source	Interrupts definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI0_STC	SPI1 Serial Transfer Complete
19	0x0024	USART0_RX	USART0 Rx Complete
20	0x0026	USART0_UDRE	USART0, Data Register Empty
21	0x0028	USART0_TX	USART0, Tx Complete
22	0x002A	ADC	ADC Conversion Complete



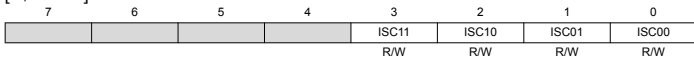
# Externe Interrupts: Register

## ■ Steuerregister für INT0 und INT1

- **EIMSK** **External Interrupt Mask Register:** Legt fest, ob die Quellen  $INT_i$  IRQs auslösen (Bit  $INT_i=1$ ) oder deaktiviert sind (Bit  $INT_i=0$ ) [1, S. 84]



- **EICRA** **External Interrupt Control Register A:** Legt für externe Interrupts INT0 und INT1 fest, wodurch ein IRQ ausgelöst wird (Flanken-/Pegelsteuerung) [1, S. 83]



Jeweils zwei *Interrupt-Sense-Control*-Bits ( $ISC_i0$  und  $ISC_i1$ ) steuern dabei die Auslöser (Tabelle für INT1, für INT0 gilt entsprechendes):

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.



- **Schritt 1:** Installation der **Interrupt-Service-Routine**
  - ISR in Hochsprache  $\rightsquigarrow$  Registerinhalte sichern und wiederherstellen
  - Unterstützung durch die **avrlibc**: Makro **ISR(SOURCE\_vect)** (Modul **avr/interrupt.h**)

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

ISR(INT1_vect) { // invoked for every INT1 IRQ
    static uint8_t counter = 0;
    sb_7seg_showNumber(counter++);
    if (counter == 100) counter = 0;
}

void main(void) {
    ... // setup
}
```



## ■ Schritt 2: Konfigurieren der Interrupt-Steuerung

- Steuerregister dem Wunsch entsprechend initialisieren
- Unterstützung durch die `avrlibc`: Makros für Bit-Indizes (Modul `avr/interrupt.h` und `avr/io.h`)

```
...  
void main(void) {  
    DDRD  &= ~(1<<PD3);           // PD3: input with pull-up  
    PORTD |= (1<<PD3);  
    EICRA &= ~(1<<ISC10 | 1<<ISC11); // INT1: IRQ on level=low  
    EIMSK |= (1<<INT1);           // INT1: enable  
    ...  
    sei();                         // global IRQ enable  
    ...  
}
```

## ■ Schritt 3: Interrupts global zulassen

- Nach Abschluss der Geräteinitialisierung
- Unterstützung durch die `avrlibc`: Befehl `sei()` (Modul `avr/interrupt.h`)



- **Schritt 4:** Wenn nichts zu tun, den **Stromsparmmodus betreten**
  - Die `sleep`-Instruktion hält die CPU an, bis ein IRQ eintrifft
    - In diesem Zustand wird nur sehr wenig Strom verbraucht
  - Unterstützung durch die `avrlibc` (Modul `avr/sleep.h`):
    - `sleep_enable()` / `sleep_disable()`: Sleep-Modus erlauben / verbieten
    - `sleep_cpu()`: Sleep-Modus betreten



```
#include <avr/sleep.h>
...
void main(void) {
    ...
    sei(); // global IRQ enable
    while(1) {
        sleep_enable();
        sleep_cpu(); // wait for IRQ
        sleep_disable();
    }
}
```

