

# Angewandte Informatik

## Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

27. November 2017

# Angewandte Informatik

## Hardwarenahe Programmierung

<https://gitlab.cvh-server.de/pgerwinski/hp.git>

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Hardwarenahe Programmierung
  - 4.1 Bit-Operationen
  - 4.2 I/O-Ports
  - 4.3 Interrupts
  - 4.4 volatile-Variable
  - 4.5 Byte-Reihenfolge – Endianness
  - 4.6 Speicherausrichtung – Alignment
- 5 Algorithmen
  - 5.1 Differentialgleichungen
  - ...

...

## 4 Hardwarenahe Programmierung

### 4.1 Bit-Operationen

#### 4.1.1 Zahlensysteme

Basis	Name	Beispiel	Anwendung
2	Binärsystem	1 0000 0011	Bit-Operationen
8	Oktalsystem	0403	Dateizugriffsrechte (Unix)
10	Dezimalsystem	259	Alltag
16	Hexadezimalsystem	0x103	Bit-Operationen
256	(keiner gebräuchlich)	0.0.1.3	IP-Adressen (IPv4)

### 4.1.1 Zahlensysteme

Oktal- und Hexadezimal-Zahlen lassen sich ziffernweise in Binär-Zahlen umrechnen:

000	0	0000	0	1000	8
001	1	0001	1	1001	9
010	2	0010	2	1010	A
011	3	0011	3	1011	B
100	4	0100	4	1100	C
101	5	0101	5	1101	D
110	6	0110	6	1110	E
111	7	0111	7	1111	F

## 4.1.1 Zahlensysteme

$$\begin{array}{r} \text{AFFE} \\ + \quad 1 \\ \hline \text{AFFF} \end{array}$$

$$\begin{array}{r} \text{AFFE} \\ + \quad 2 \\ \hline \text{B000} \end{array}$$

$$\begin{array}{r} 6789 \\ + 7654 \\ \hline \text{DDDD} \end{array}$$

$$\begin{array}{r} 68AC \\ + 7654 \\ \hline \text{DE00} \end{array}$$

$$\begin{array}{r} 1011 \\ \& 0101 \\ \hline 0001 \end{array} \quad \begin{array}{r} 1011 \\ | 0101 \\ \hline 1111 \end{array} \quad \begin{array}{r} 1011 \\ \wedge 0101 \\ \hline 1110 \end{array}$$

$$a = 011011011100$$

$$a \mid = 000000100000 \leftarrow \text{"Maske"}$$

$$011011111100$$

$$a \& = 111111011111$$

$$011011011100$$

$$a \wedge = 000000000010$$

$$011011011110$$

$$\begin{array}{r} 01100011 \\ \swarrow \searrow \swarrow \searrow \swarrow \searrow \swarrow \searrow \\ 10001100 \end{array}$$

Maske  
generieren:  
 $1 \ll 3$

"Bit Nr. 3"

$$\begin{aligned} &= 00001000 \\ \sim(1 \ll 3) &= 11110111 \end{aligned}$$

## 4.1.2 Bit-Operationen in C

C-Operator	Verknüpfung	Anwendung
&	Und	Bits gezielt löschen
	Oder	Bits gezielt setzen
^	Exklusiv-Oder	Bits gezielt invertieren
~	Nicht	Alle Bits invertieren
<<	Verschiebung nach links	Maske generieren
>>	Verschiebung nach rechts	Bits isolieren

Numerierung der Bits: von rechts ab 0

Bit Nr. 3 auf 1 setzen: `a |= 1 << 3;`

Bit Nr. 4 auf 0 setzen: `a &= ~(1 << 4);`

Bit Nr. 0 invertieren: `a ^= 1 << 0;`

Abfrage, ob Bit Nr. 1 gesetzt ist: `if (a & (1 << 1))`

## 4.1.2 Bit-Operationen in C

C-Datentypen für Bit-Operationen:

**#include** <stdint.h>

	8 Bit	16 Bit	32 Bit	64 Bit
mit Vorzeichen	int8_t	int16_t	int32_t	int64_t
ohne Vorzeichen	uint8_t	uint16_t	uint32_t	uint64_t

Ausgabe:

**#include** <stdio.h>

**#include** <stdint.h>

**#include** <inttypes.h>

...

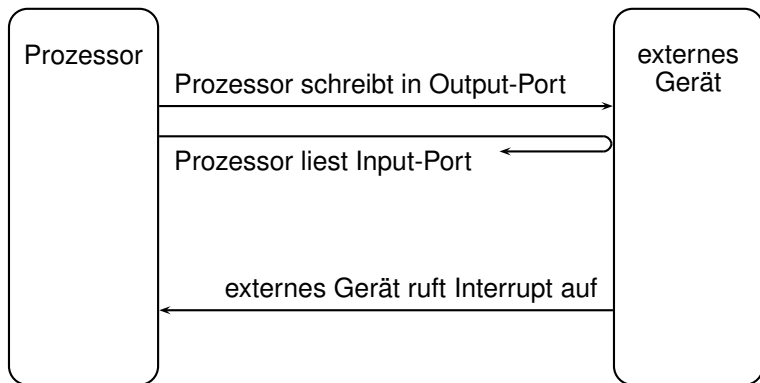
uint64\_t x = 42;

printf ("Die\_Antwort\_lautet:\_% " PRIu64 "\n", x);

## 4.2 I/O-Ports

## 4.3 Interrupts

Kommunikation mit externen Geräten





## 4.2 I/O-Ports

In Output-Port schreiben = Aktoren ansteuern

Beispiel: LED

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

```
...
```

```
DDRC = 0x70;    binär: 0111 0000
```

```
PORTC = 0x40;   binär: 0100 0000
```

Herstellerspezifisch!

**DDR** = Data Direction Register

Bit = 1 für Output-Port

Bit = 0 für Input-Port

*Details: siehe Datenblatt und Schaltplan*

## 4.2 I/O-Ports

Aus Input-Port lesen = Sensoren abfragen

Beispiel: Taster

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

```
...
```

```
DDRC = 0xfd;          binär: 1111 1101
```

```
while (PINC & 0x02 == 0) binär: 0000 0010
```

```
; /* just wait */
```

Herstellerspezifisch!

DDR = Data Direction Register

Bit = 1 für Output-Port

Bit = 0 für Input-Port

*Details: siehe Datenblatt und Schaltplan*

Praktikumsaufgabe: Druckknopfampel

## 4.3 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: eingebaute Uhr

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

```
...
```

„Dies ist ein Interrupt-Handler.“

Interrupt-Vektor darauf zeigen lassen

```
ISR (TIMER0B_COMP_vect)
{
    PORTD ^= 0x40;
}
```

Herstellerspezifisch!

Initialisierung über spezielle Ports: `TCCR0B`, `TIMSK0`

*Details: siehe Datenblatt und Schaltplan*

## 4.3 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: eingebaute Uhr

statt Zählschleife (`_delay_ms`):  
Hauptprogramm kann  
andere Dinge tun

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

... „Dies ist ein Interrupt-Handler.“  
Interrupt-Vektor darauf zeigen lassen

```
ISR (TIMER0B_COMP_vect)
{
    PORTD ^= 0x40;
}
```

Herstellerspezifisch!

Initialisierung über spezielle Ports: `TCCR0B`, `TIMSK0`

*Details: siehe Datenblatt und Schaltplan*

## 4.3 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

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

```
...
```

```
ISR (INT0_vect)
```

```
{  
    PORTD ^= 0x40;  
}
```

statt *Busy Waiting*:  
Hauptprogramm kann  
andere Dinge tun

Herstellerspezifisch!

Initialisierung über spezielle Ports: `EICRA`, `EIMSK`

*Details: siehe Datenblatt und Schaltplan*

## 4.4 volatile-Variable

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“  
Beispiel: Taster

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

```
...
```

```
uint8_t key_pressed = 0;
```

```
ISR (INT0_vect)
```

```
{  
    key_pressed = 1;  
}
```

```
int main (void)
```

```
{
```

```
...
```

```
while (1)
```

```
{
```

```
    while (!key_pressed)
```

```
        ; /* just wait */
```

```
    PORTD ^= 0x40;
```

```
    key_pressed = 0;
```

```
}
```

```
return 0;
```

```
}
```

## 4.4 volatile-Variable

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“  
Beispiel: Taster

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

```
...
```

```
volatile uint8_t key_pressed = 0;
```

```
ISR (INT0_vect)
```

```
{  
    key_pressed = 1;  
}
```

```
int main (void)
```

```
{
```

```
...
```

```
while (1)
```

```
{
```

```
    while (!key_pressed)
```

```
        ; /* just wait */
```

```
        PORTD ^= 0x40;
```


```
        key_pressed = 0;
```

```
    }
```

```
    return 0;
```

```
}
```

**volatile:**  
Speicherzugriff  
nicht wegoptimieren



# Angewandte Informatik

## Hardwarenahe Programmierung

<https://gitlab.cvh-server.de/pgerwinski/hp.git>

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Hardwarenahe Programmierung
  - 4.1 Bit-Operationen
  - 4.2 I/O-Ports
  - 4.3 Interrupts
  - 4.4 volatile-Variable
  - 4.5 Byte-Reihenfolge – Endianness
  - 4.6 Speicherausrichtung – Alignment
- 5 Algorithmen
  - 5.1 Differentialgleichungen
  - ...

...