

# Angewandte Informatik

## Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

5. Dezember 2016

# Angewandte Informatik

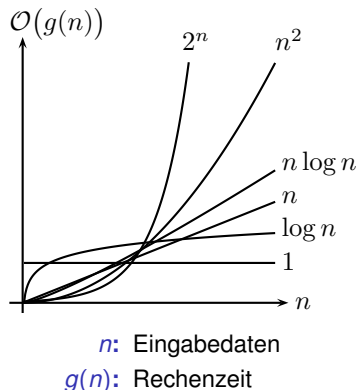
## Hardwarenahe Programmierung

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Algorithmen
  - 4.1 Differentialgleichungen
  - 4.2 Rekursion
  - 4.3 Aufwandsabschätzungen
- 5 Hardwarenahe Programmierung
  - 5.1 Bit-Operationen
  - 5.2 I/O-Ports
  - 5.3 Interrupts
  - 5.4 volatile-Variable
  - ...
- 6 Objektorientierte Programmierung
- ...

## 4.3 Aufwandsabschätzungen – Komplexitätsanalyse

### Beispiel: Sortieralgorithmen

- Minimum suchen:  $\mathcal{O}(n)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen  
→ Selectionsort:  $\mathcal{O}(n^2)$
- Während Minimumsuche prüfen und abbrechen, falls schon sortiert  
→ Bubblesort:  $\mathcal{O}(n)$  bis  $\mathcal{O}(n^2)$
- Rekursiv sortieren  
→ Quicksort:  $\mathcal{O}(n \log n)$  bis  $\mathcal{O}(n^2)$



# 5 Hardwarenahe Programmierung

## 5.1 Bit-Operationen

### 5.1.1 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)

## 5.1.1 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

## 5.1.2 Bit-Operationen in C

C-Operator	Verknüpfung	Anwendung
<code>&amp;</code>	Und	Bits gezielt löschen
<code> </code>	Oder	Bits gezielt setzen
<code>^</code>	Exklusiv-Oder	Bits gezielt invertieren
<code>~</code>	Nicht	Alle Bits invertieren
<code>&lt;&lt;</code>	Verschiebung nach links	Maske generieren
<code>&gt;&gt;</code>	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))`

## 5.1.2 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);

d: dezimal, mit Vorzeichen

u: dezimal, ohne Vorzeichen

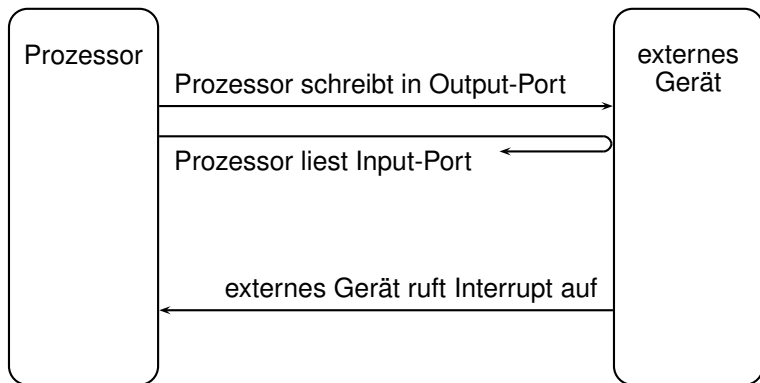
x: hexadezimal, ohne Vorzeichen

C-String-Konstante zusammenfügen  
durch Hintereinanderschreiben

## 5.2 I/O-Ports

## 5.3 Interrupts

Kommunikation mit externen Geräten





## 5.2 I/O-Ports

In Output-Port schreiben = Aktoren ansteuern

Beispiel: LED

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

```
...
```

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

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

Herstellerspezifisch!

*Details: siehe Datenblatt und Schaltplan*

## 5.2 I/O-Ports

Programm `blink.c` compilieren:

```
$ avr-gcc -Wall -Os -c -mmcu=atmega328p blink.c \  
-o blink.elf
```

ausführbare Datei in Speicherabbild umwandeln:

```
$ avr-objcopy -O ihex blink.elf blink.hex
```

auf Mikro-Controller aufspielen („herunterladen“):

```
$ avrdude -P /dev/ttyACM0 -c arduino -p m16 \  
-U flash:w:blink.hex
```

## 5.2 I/O-Ports

Aus Input-Port lesen = Sensoren abfragen

Beispiel: Taster

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

```
...
```

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

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

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

Herstellerspezifisch!

*Details: siehe Datenblatt und Schaltplan*

Praktikumsaufgabe: Druckknopfampel

## 5.2 I/O-Ports

Aus Input-Port lesen = Sensoren abfragen

Beispiel: Taster

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

```
...
```

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

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

```
; /* just wait */ ← Busy Waiting
```

Herstellerspezifisch!

*Details: siehe Datenblatt und Schaltplan*

Praktikumsaufgabe: Druckknopfampel

## 5.3 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*

## 5.3 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>
```

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

„Dies ist ein Interrupt-Handler.“

Interrupt-Vektor darauf zeigen lassen

Herstellerspezifisch!

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

*Details: siehe Datenblatt und Schaltplan*

## 5.3 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*

## 5.4 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;
```

```
}
```



## 5.4 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

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Algorithmen
- 5 Hardwarenahe Programmierung
  - 5.1 Bit-Operationen
  - 5.2 I/O-Ports
  - 5.3 Interrupts
  - 5.4 volatile-Variable
  - 5.5 Byte-Reihenfolge – Endianness
  - 5.6 Speicherausrichtung – Alignment
- 6 Objektorientierte Programmierung

...

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.  
Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen  
Welche Bits liegen wo?

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

 Big-Endian „großes Ende zuerst“

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

Big-Endian „großes Ende zuerst“  
für Menschen leichter lesbar

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

 Big-Endian „großes Ende zuerst“  
für Menschen leichter lesbar

03	04
----	----

 Little-Endian „kleines Ende zuerst“

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

 Big-Endian „großes Ende zuerst“  
für Menschen leichter lesbar

03	04
----	----

 Little-Endian „kleines Ende zuerst“  
bei Additionen effizienter



## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

Big-Endian „großes Ende zuerst“  
für Menschen leichter lesbar

03	04
----	----

Little-Endian „kleines Ende zuerst“  
bei Additionen effizienter

→ Geschmackssache

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.  
Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen  
Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

 Big-Endian „großes Ende zuerst“  
für Menschen leichter lesbar

03	04
----	----

 Little-Endian „kleines Ende zuerst“  
bei Additionen effizienter

→ Geschmackssache  
... **außer bei Datenaustausch!**

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.  
Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

—→ Geschmackssache

... **außer bei Datenaustausch!**

- Dateiformate
- Datenübertragung

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.2 Dateiformate

Audio-Formate: Reihenfolge der Bytes in 16- und 32-Bit-Zahlen

- RIFF-WAVE-Dateien (.wav): Little-Endian
- Au-Dateien (.au): Big-Endian

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.2 Dateiformate

Audio-Formate: Reihenfolge der Bytes in 16- und 32-Bit-Zahlen

- RIFF-WAVE-Dateien (.wav): Little-Endian
- Au-Dateien (.au): Big-Endian
- ältere AIFF-Dateien (.aiff): Big-Endian
- neuere AIFF-Dateien (.aiff): Little-Endian

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.2 Dateiformate

Audio-Formate: Reihenfolge der Bytes in 16- und 32-Bit-Zahlen

- RIFF-WAVE-Dateien (.wav): Little-Endian
- Au-Dateien (.au): Big-Endian
- ältere AIFF-Dateien (.aiff): Big-Endian
- neuere AIFF-Dateien (.aiff): Little-Endian

Grafik-Formate: Reihenfolge der Bits in den Bytes

- PBM-Dateien: Big-Endian
- XBM-Dateien: Little-Endian

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.2 Dateiformate

Audio-Formate: Reihenfolge der Bytes in 16- und 32-Bit-Zahlen

- RIFF-WAVE-Dateien (.wav): Little-Endian
- Au-Dateien (.au): Big-Endian
- ältere AIFF-Dateien (.aiff): Big-Endian
- neuere AIFF-Dateien (.aiff): Little-Endian

Grafik-Formate: Reihenfolge der Bits in den Bytes

- PBM-Dateien: Big-Endian, MSB first
- XBM-Dateien: Little-Endian, LSB first

MSB/LSB = most/least significant bit

## 5.5 Byte-Reihenfolge – Endianness

### 5.5.3 Datenübertragung

- RS-232 (serielle Schnittstelle): LSB first
- I<sup>2</sup>C: MSB first
- USB: beides



## 5.5 Byte-Reihenfolge – Endianness

### 5.5.3 Datenübertragung

- RS-232 (serielle Schnittstelle): LSB first
- I<sup>2</sup>C: MSB first
- USB: beides
- Ethernet: LSB first
- TCP/IP (Internet): Big-Endian

# Angewandte Informatik

## Hardwarenahe Programmierung

- 1 Einführung**
- 2 Einführung in C**
- 3 Bibliotheken**
- 4 Algorithmen**
- 5 Hardwarenahe Programmierung**
  - 5.1** Bit-Operationen
  - 5.2** I/O-Ports
  - 5.3** Interrupts
  - 5.4** volatile-Variable
  - 5.5** Byte-Reihenfolge – Endianness
  - 5.6** Speicherausrichtung – Alignment
- 6 Objektorientierte Programmierung**

...