

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

23. November 2023

Hardwarenahe Programmierung

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

- 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 Binärdarstellung negativer Zahlen
 - 4.7 Binärdarstellung von Gleitkommazahlen
 - 4.8 Speicherausrichtung – Alignment
- 5 Algorithmen**
- 6 Objektorientierte Programmierung**
- 7 Datenstrukturen**

4.5 Byte-Reihenfolge – Endianness

4.5.1 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!**

4.5 Byte-Reihenfolge – Endianness

4.5.1 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

4.5 Byte-Reihenfolge – Endianness

4.5.2 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

4.5 Byte-Reihenfolge – Endianness

4.5.3 Datenübertragung

- RS-232 (serielle Schnittstelle): LSB first
- I²C: MSB first
- USB: beides
- Ethernet: LSB first
- TCP/IP (Internet): Big-Endian

4.6 Binärdarstellung negativer Zahlen

Speicher ist begrenzt!

→ feste Anzahl von Bits

8-Bit-Zahlen ohne Vorzeichen: `uint8_t`

→ Zahlenwerte von `0x00` bis `0xff` = 0 bis 255

→ $255 + 1 = 0$

8-Bit-Zahlen mit Vorzeichen: `int8_t`

`0xff` = 255 ist die „natürliche“ Schreibweise für -1 .

→ Zweierkomplement

Oberstes Bit = 1: negativ

Oberstes Bit = 0: positiv

→ $127 + 1 = -128$

4.6 Binärdarstellung negativer Zahlen

Speicher ist begrenzt!

→ feste Anzahl von Bits

16-Bit-Zahlen ohne Vorzeichen: `uint16_t`

→ Zahlenwerte von `0x0000` bis `0xffff` = 0 bis 65535

→ $65535 + 1 = 0$

`uint8_t`

0 bis 255

$255 + 1 = 0$

16-Bit-Zahlen mit Vorzeichen: `int16_t`

`0xffff` = 65535 ist die „natürliche“ Schreibweise für -1 .

→ Zweierkomplement

`int8_t`

`0xff` = 255 = -1

Oberstes Bit = 1: negativ

Oberstes Bit = 0: positiv

→ $32767 + 1 = -32768$

Literatur: <http://xkcd.com/571/>

4.6 Binärdarstellung negativer Zahlen

Frage: Für welche Zahl steht der Speicherinhalt

| | |
|----|----|
| a3 | 90 |
|----|----|

 (hexadezimal)?

Antwort: Das kommt darauf an. ;–)

Little-Endian:

| | | |
|-----------------------------------|--------|---|
| als <code>int8_t</code> : | –93 | (nur erstes Byte) |
| als <code>uint8_t</code> : | 163 | (nur erstes Byte) |
| als <code>int16_t</code> : | –28509 | |
| als <code>uint16_t</code> : | 37027 | |
| <code>int32_t</code> oder größer: | 37027 | (zusätzliche Bytes mit Nullen aufgefüllt) |

Big-Endian:

| | | |
|-----------------------------|----------------------|---|
| als <code>int8_t</code> : | –93 | (nur erstes Byte) |
| als <code>uint8_t</code> : | 163 | (nur erstes Byte) |
| als <code>int16_t</code> : | –23664 | |
| als <code>uint16_t</code> : | 41872 | |
| als <code>int32_t</code> : | –1550843904 | (zusätzliche Bytes mit Nullen aufgefüllt) |
| als <code>uint32_t</code> : | 2744123392 | |
| als <code>int64_t</code> : | –6660823848880963584 | |
| als <code>uint64_t</code> : | 11785920224828588032 | |

4.7 Binärdarstellung von Gleitkommazahlen

Beispiel für Gleitkommazahl: $2,351 \cdot 10^5$ (oder: 2.351×10^5)

Bezeichnungen: Mantisse $\cdot 10^{\text{Exponent}}$

C-Schreibweise: **2.351e5** (oder: **2.351E5**)

Wie speichert man Gleitkommazahlen?

m -Bit-Zahl, davon

- e Bits für den Exponenten (einschließlich Vorzeichen),
- 1 Bit für das Vorzeichen der Mantisse,
- $m - e - 1$ Bits für die Mantisse.

4.7 Binärdarstellung von Gleitkommazahlen

Beispiel für Gleitkommazahl: $2,351 \cdot 10^5$ (oder: 2.351×10^5)

Bezeichnungen: Mantisse $\cdot 10^{\text{Exponent}}$

C-Schreibweise: **2.351e5** (oder: **2.351E5**)

Wie speichert man Gleitkommazahlen?

m -Bit-Zahl, davon

- e Bits für den Exponenten (einschließlich Vorzeichen),
- 1 Bit für das Vorzeichen der Mantisse,
- $m - e - 1$ Bits für die Mantisse.

Trick: Mantisse als *normalisierte Zahl* abspeichern

Vorteil gegenüber ganzen Zahlen:

größerer Wertebereich bei vergleichbarem Speicherplatzbedarf

Nachteil gegenüber ganzen Zahlen: Rundungsfehler

→ **ungeeignet** für Anwendungen, bei denen es auf jedes Bit ankommt
(z. B. Verschlüsselung)

4.7 Binärdarstellung von Gleitkommazahlen

Problem beim Arbeiten mit Gleitkommazahlen: Auslöschung von Ziffern

- Zahlen aufsummieren:
vorher sortieren, mit der kleinsten Zahl beginnen

4.7 Binärdarstellung von Gleitkommazahlen

Problem beim Arbeiten mit Gleitkommazahlen: Auslöschung von Ziffern

- Zahlen aufsummieren:
vorher sortieren, mit der kleinsten Zahl beginnen
- Ableitungen bilden:
Beim Bilden von Differenzquotienten
verliert man notwendigerweise an Präzision!
→ Die Differenzen sehr sorgfältig auswählen.
→ Am besten gar nicht ableiten, sondern integrieren.

4.7 Binärdarstellung von Gleitkommazahlen

Problem beim Arbeiten mit Gleitkommazahlen: Auslöschung von Ziffern

- Zahlen aufsummieren:
vorher sortieren, mit der kleinsten Zahl beginnen
- Ableitungen bilden:
Beim Bilden von Differenzquotienten
verliert man notwendigerweise an Präzision!
→ Die Differenzen sehr sorgfältig auswählen.
→ Am besten gar nicht ableiten, sondern integrieren.
(Trick: Ableiten über Fourier-Transformationen)

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter
- ... oder sogar nur dann erlaubt

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter
- ... oder sogar nur dann erlaubt

→ Compiler optimiert Speicherausrichtung

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter
- ... oder sogar nur dann erlaubt

→ Compiler optimiert Speicherausrichtung

```
uint8_t a;  
uint8_t dummy;  
uint16_t b;  
uint8_t c;
```

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter
- ... oder sogar nur dann erlaubt

→ Compiler optimiert Speicherausrichtung

```
uint8_t a;  
uint8_t dummy;    uint8_t a;  
uint16_t b;        uint8_t c;  
uint8_t c;         uint16_t b;
```

4.8 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter
- ... oder sogar nur dann erlaubt

→ Compiler optimiert Speicherausrichtung

```
uint8_t a;  
uint8_t dummy;  
uint16_t b;  
uint8_t c;  
  
uint8_t a;  
uint8_t c;  
uint16_t b;
```

Fazit:

- **Adressen von Variablen sind systemabhängig**
- Bei Definition von Datenformaten Alignment beachten → effizienter

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t)$$

$$y'(t) = v_y(t)$$

$$v'_x(t) = 0$$

$$v'_y(t) = -g$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \qquad x(t) = \int v_x(t) dt$$

$$y'(t) = v_y(t) \qquad \Rightarrow \qquad y(t) = \int v_y(t) dt$$

$$v'_x(t) = 0 \qquad v_x(t) = \int 0 dt$$

$$v'_y(t) = -g \qquad v_y(t) = \int -g dt$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \qquad x(t) = \int v_x(t) dt$$

$$y'(t) = v_y(t) \qquad y(t) = \int v_y(t) dt$$

\Rightarrow

$$v'_x(t) = 0 \qquad v_x(t) = \int 0 dt = v_{0x}$$

$$v'_y(t) = -g \qquad v_y(t) = \int -g dt$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \qquad x(t) = \int v_x(t) dt = \int v_{0x} dt$$

$$y'(t) = v_y(t) \qquad y(t) = \int v_y(t) dt$$

\Rightarrow

$$v'_x(t) = 0 \qquad v_x(t) = \int 0 dt = v_{0x}$$

$$v'_y(t) = -g \qquad v_y(t) = \int -g dt$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \qquad x(t) = \int v_x(t) dt = \int v_{0x} dt = x_0 + v_{0x} \cdot t$$

$$y'(t) = v_y(t) \qquad y(t) = \int v_y(t) dt$$

\Rightarrow

$$v'_x(t) = 0 \qquad v_x(t) = \int 0 dt = v_{0x}$$

$$v'_y(t) = -g \qquad v_y(t) = \int -g dt$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \qquad x(t) = \int v_x(t) dt = \int v_{0x} dt = x_0 + v_{0x} \cdot t$$

$$y'(t) = v_y(t) \qquad y(t) = \int v_y(t) dt$$

\Rightarrow

$$v'_x(t) = 0 \qquad v_x(t) = \int 0 dt = v_{0x}$$

$$v'_y(t) = -g \qquad v_y(t) = \int -g dt = v_{0y} - g \cdot t$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \qquad x(t) = \int v_x(t) dt = \int v_{0x} dt = x_0 + v_{0x} \cdot t$$

$$y'(t) = v_y(t) \qquad \Rightarrow \qquad y(t) = \int v_y(t) dt = \int v_{0y} - g \cdot t dt$$

$$v'_x(t) = 0 \qquad v_x(t) = \int 0 dt = v_{0x}$$

$$v'_y(t) = -g \qquad v_y(t) = \int -g dt = v_{0y} - g \cdot t$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \quad x(t) = \int v_x(t) dt = \int v_{0x} dt = x_0 + v_{0x} \cdot t$$

$$y'(t) = v_y(t) \quad \Rightarrow \quad y(t) = \int v_y(t) dt = \int v_{0y} - g \cdot t dt = y_0 + v_{0y} \cdot t - \frac{1}{2}gt^2$$

$$v'_x(t) = 0 \quad v_x(t) = \int 0 dt = v_{0x}$$

$$v'_y(t) = -g \quad v_y(t) = \int -g dt = v_{0y} - g \cdot t$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

$$x'(t) = v_x(t) \quad x \mathrel{+=} v_x * dt;$$

$$y'(t) = v_y(t) \quad y \mathrel{+=} v_y * dt;$$

\Rightarrow

$$v'_x(t) = 0 \quad v_x \mathrel{+=} 0 * dt;$$

$$v'_y(t) = -g \quad v_y \mathrel{+=} -g * dt;$$

5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

Beispiel 2: Mathematisches Pendel

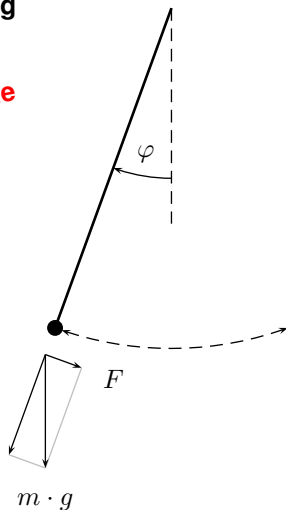
$$\varphi'(t) = \omega(t)$$

$$\omega'(t) = -\frac{g}{l} \cdot \sin \varphi(t)$$

→ nächste Woche

- Von Hand (analytisch):
Lösung raten (Ansatz), Parameter berechnen
- Mit Computer (numerisch):
Eulersches Polygonzugverfahren

```
phi += dt * omega;  
omega += - dt * g / l * sin (phi);
```



5 Algorithmen

5.1 Differentialgleichungen

Beispiel 1: Gleichmäßig beschleunigte Bewegung

Beispiel 2: Mathematisches Pendel

$$\varphi'(t) = \omega(t)$$

$$\omega'(t) = -\frac{g}{l} \cdot \sin \varphi(t)$$

→ nächste Woche

- Von Hand (analytisch):
Lösung raten (Ansatz), Parameter berechnen
- Mit Computer (numerisch):
Eulersches Polygonzugverfahren

```
phi += dt * omega;  
omega += - dt * g / l * sin (phi);
```

Beispiel 3: Weltraum-Simulation

Praktikumsaufgabe

