

Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2024

Übung 2

Maxim Ritter von Oncuil
Arne Vogel

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Informatik 4
Systemsoftware



Friedrich-Alexander-Univ
Technische Fakultät

Verwendung von int



- Die Größe von `int` ist nicht genau definiert
 - Zum Beispiel beim ATMEGA328PB: 16 bit
 - ⇒ Gerade auf μC führt dies zu langsamerem Code und/oder Fehlern
 - Für die Übung gilt
 - Verwendung von `int` ist ein Fehler
 - Stattdessen: Verwendung der in der `stdint.h` definierten Typen: `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, etc.
 - Wertebereich
 - `limits.h`: `INT8_MAX`, `INT8_MIN`, ...
 - Speicherplatz ist auf μC sehr teuer (SPiCBOARD/ATMEGA328PB hat nur 2048 Byte SRAM)
- ↪ Nur so viel Speicher verwenden, wie tatsächlich benötigt wird!

Variablen

Typedefs & Enums



```
01 #define PB3 3
02
03 typedef enum {
04     BUTTON0 = 0, BUTTON1 = 1
05 } BUTTON;
06
07 typedef enum {
08     PRESSED = 0, RELEASED = 1, UNKNOWN = 2
09 } BUTTONSTATE;
10
11 void main(void) {
12     /* ... */
13     PORTB |= (1 << PB3); // nicht (1 << 3)
14
15     // Deklaration: BUTTONSTATE sb_button_getState(BUTTON btn);
16     BUTTONSTATE zustand = sb_button_getState(BUTTON0); // nicht
17     ↪ sb_button_getState(0)
18     /* ... */
19 }
```

- Vordefinierte Typen verwenden
- Explizite Zahlenwerte nur verwenden, wenn notwendig



Bits & Bytes

- Zahlen können in unterschiedlichen Basen dargestellt werden
⇒ Üblich: dezimal (10), hexadezimal (16), oktal (8) und binär (2)
- Nomenklatur:
 - Bits: Ziffern von Binärzahlen
 - Nibbles: Gruppen von 4 Bits
 - Bytes: Gruppen von 8 Bits

3

Bitoperationen



Bitoperationen



- Bitoperation: Bitweise logische Verknüpfung
- Mögliche Operationen:

~		&	0	1			0	1		^	0	1	
0	1	0	0	0	0	0	0	0	1	0	0	1	1
1	0	1	0	1	1	1	1	1	1	1	1	0	0
nicht		und		oder		exklusives oder							

- Bitoperation: Bitweise logische Verknüpfung
- Mögliche Operationen:

~		&	0	1			0	1		^	0	1	
0	1	0	0	0	0	0	0	0	1	0	0	1	1
1	0	1	0	1	1	1	1	1	1	1	1	0	0
nicht		und		oder		exklusives oder							

- Beispiel:

	1100 ₂	1100 ₂	1100 ₂
~ 1001 ₂	& 1001 ₂	1001 ₂	^ 1001 ₂
0110 ₂	1000 ₂	1101 ₂	0101 ₂



- Beispiel:

```
uint8_t x = 0x9d;
x = x << 2;
x = x >> 2;
```

1	0	0	1	1	1	0	1
0	1	1	1	0	1	0	0
0	0	0	1	1	1	0	1

- Setzen von Bits:

```
(1 << 0)
(1 << 3)
(1 << 3) | (1 << 0)
```

0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	1

- **Achtung:**

Bei signed-Variablen ist das Verhalten des >>-Operators nicht vollständig definiert. In der Regel werden bei negativen Werten 1er geschiftet.

Aufgabe: snake

5

Aufgabe: snake



Allgemeine Hinweise



- Schlange bestehend aus benachbarten LEDs
- Länge 1 bis 5 LEDs, regelbar mit Potentiometer (POTI)
- Geschwindigkeit abhängig von der Umgebungshelligkeit (PHOTO)
 - ~> Je heller die Umgebung, desto schneller
- Modus der Schlange mit Taster (BUTTON0) umschaltbar
 - Normal: Leuchtende LEDs repräsentieren Schlange
 - Invertiert: Inaktive LEDs repräsentieren Schlange

⇒ Bearbeitung in Zweiergruppen: submit fragt nach Partner

- Variablen in Funktionen verhalten sich weitgehend wie in Java
 - ~> Zur Lösung der Aufgabe sind lokale Variablen ausreichend
- Der C-Compiler liest Dateien von oben nach unten
 - ~> Legen Sie die Funktionen in der folgenden Reihenfolge an:
 1. wait()
 2. drawsake()
 3. main()

⇒ Details zum Kompilieren werden in der Vorlesung besprochen.

6

7



- Position des Kopfes
 - Nummer einer LED
 - Wertebereich $\{0, 1, \dots, 7\}$
 - Länge der Schlange
 - Ganzzahl aus $\{1, 2, \dots, 5\}$
 - Modus der Schlange
 - Hell oder dunkel
 - Beispielsweise durch 0 und 1 repräsentiert
 - Geschwindigkeit der Schlange
 - Hier: Durchlaufzahl der Warteschleife
- Basisablauf: Welche Schritte wiederholen sich immer wieder?
 - Vermeidung von Codeduplikation:
 - ~ Wiederkehrende Teilprobleme in eigene Funktionen auslagern
 - Kapselung: Sichtbarkeit möglichst weit einschränken
 - Ist der Zustand nur für eine Funktion relevant?
 - ~ Lokale Variable
 - Greifen mehrere Funktionen auf den gleichen Zustand zu?
 - ~ Modullokal/global Variable

8

9



- Basisablauf: Schlange darstellen, Schlange bewegen, ...
- Pseudocode:

```

01 void main(void) {
02     while(1) {
03         // Berechne Laenge
04         laenge = ...
05
06         // Zeichne Schlange
07         drawSnake(kopf, laenge, modus);
08
09         // Setze Schlängenkopf weiter
10         ...
11
12         // Warte und bestimme Modus
13         ...
14
15     } // Ende der Hauptschleife
16 }

```

- Darstellungsparameter
 - Kopfposition
 - Länge
 - Modus
- Funktionssignatur:


```
void drawSnake(uint8_t head, uint8_t length,
               ↪ uint8_t modus)
```
- Anzeige der Schlange abhängig von den Parametern
 - Normaler Modus (Helle Schlange):
 - Aktivieren der zur Schlange gehörenden LEDs
 - Deaktivieren der restlichen LEDs
 - Invertierter Modus (Dunkle Schlange):
 - Deaktivieren der zur Schlange gehörenden LEDs
 - Aktivieren der restlichen LEDs

10

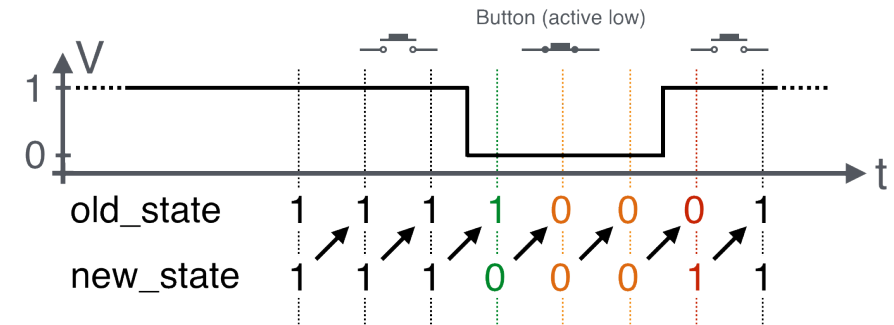
11



- Bewegen der Schlange
 - Kopfposition abhängig von der Bewegungsrichtung anpassen
 - Problem: Was passiert am Ende der LED-Leiste?
- Eine Lösung: Der Modulooperator %
 - Divisionsrest einer Ganzzahldivision
 - **Achtung:** In C ist das Ergebnis im negativen Bereich auch negativ
 - Beispiel: `b = a % 4;`

a	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
b	-1	0	-3	-2	-1	0	1	2	3	0	1	2

- Aktives Warten zwischen Schlangenbewegungen
 - Erkennen ob der Button gedrückt wurde
 - Detektion der Flanke durch **zyklisches Abfragen** (engl. Polling) des Pegels
 - Unterscheidung zwischen **active-high** & **active-low**
 - Später: Realisierung durch Interrupts



12

13

Hands-on: Signallampe

Screencast: <https://www.video.uni-erlangen.de/clip/id/14038>

Hands-on: Signallampe



- Morsesignale über RED0 ausgeben
- Steuerung über BUTTON1
- Nutzung der Bibliotheksfunktionen für Button und LED
- Dokumentation der Bibliothek in der SPiC IDE oder unter <https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/libapi>
- Quelltext kommentieren