Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2023

Übung 2

Maximilian Ott
Arne Vogel

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

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Variablen

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!

Typedefs & Enums

```c
#define PB3 3

typedef enum {
    BUTTON0 = 0, BUTTON1 = 1
} BUTTON;

typedef enum {
    PRESSED = 0, RELEASED = 1, UNKNOWN = 2
} BUTTONSTATE;

void main(void) {
    /* ... */
    PORTB |= (1 << PB3); // nicht (1 << 3)

    // Deklaration: BUTTONSTATE sb_button_getState(BUTTON btn);
    BUTTONSTATE zustand = sb_button_getState(BUTTON0); // nicht
     ↪ sb_button_getState(0)
    /* ... */
}
```

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

Bits & Bytes

Zahlensysteme

- 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

Bitoperationen

- Bitoperation: Bitweise logische Verknüpfung
- Mögliche Operationen:

~	
0	1
1	0

nicht

&	0	1
0	0	0
1	0	1

und

|	0	1
0	0	1
1	1	1

oder

^	0	1
0	0	1
1	1	0

exklusives oder

Bitoperationen

- Bitoperation: Bitweise logische Verknüpfung
- Mögliche Operationen:

~	
0	1
1	0

nicht

&	0	1
0	0	0
1	0	1

und

|	0	1
0	0	1
1	1	1

oder

^	0	1
0	0	1
1	1	0

exklusives oder

- Beispiel:

$\sim 1001_2$	$1100_2 \;\&\; 1001_2$	$1100_2 \mid 1001_2$	$1100_2 \;\hat{}\; 1001_2$
0110_2	1000_2	1101_2	0101_2

- Beispiel:

`uint8_t x = 0x9d;`	1	0	0	1	1	1	0	1
`x = x << 2;`	0	1	1	1	0	1	**0**	**0**
`x = x >> 2;`	**0**	**0**	0	1	1	1	0	1

- Setzen von Bits:

`(1 << 0)`	0	0	0	0	0	0	0	**1**
`(1 << 3)`	0	0	0	0	**1**	0	0	0
`(1 << 3) | (1 << 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 geshiftet.

Aufgabe: snake

Aufgabe: snake

- 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

Allgemeine Hinweise

- 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. `drawsnake()`
    3. `main()`

⇒ Details zum Kompilieren werden in der Vorlesung besprochen.

Beschreibung der Schlange

- Position des Kopfes
  - Nummer einer LED
  - Wertebereich $\{0, 1, \ldots, 7\}$
- Länge der Schlange
  - Ganzzahl aus $\{1, 2, \ldots, 5\}$
- Modus der Schlange
  - Hell oder dunkel
  - Beispielsweise durch 0 und 1 repräsentiert
- Geschwindigkeit der Schlange
  - Hier: Durchlaufzahl der Warteschleife

Zerlegung in Teilprobleme

- 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?
    - ⇝ Modullokale/globale Variable

Basisablauf snake

- 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 Schlangenkopf weiter
10     ...
11
12     // Warte und bestimme Modus
13     ...
14
15   } // Ende der Hauptschleife
16 }
```


Darstellung der Schlange

- 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

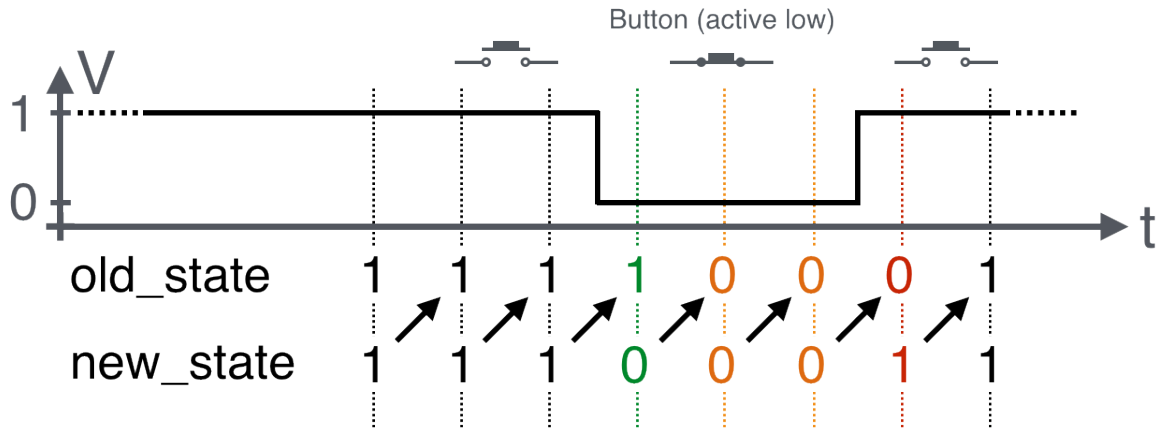
Der Modulo Operator

- 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

Flankendetektion ohne Interrupts

- 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



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/SS23/spic/uebung/spicboard/libapi
- Quelltext kommentieren