Exercises in System Level Programming (SLP) – Summer Term 2025

Exercise 2

Maxim Ritter von Onciul
Eva Dengler

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

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Faculty of Engineering

Variables

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 (SPIC board/ATmega328PB hat nur 2048 Byte SRAM)

⇝ Nur so viel Speicher verwenden, wie tatsächlich benötigt wird!

Usage von `int`

- The size of the `int` type is not defined exactly
- For example on ATmega328PB: 16 bit
  - ⇒ Especially in the context of µC, this can yield slower code and/or be a potential source for errors
- For working on the assignments, we decided
  - Usage of `int` counts as an error
  - Instead: Use types defined in `stdint.h`: `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, etc.
- Range of value
  - `limits.h`: `INT8_MAX`, `INT8_MIN`, ...
- Memory is limited and therefore expensive on µC (SPIC BOARD/ATmega328PB only has 2048 byte SRAM)

⇝ Only use as little memory as necessary!

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
        ↪ sb_button_getState(0)
17     /* ... */
18 }
```

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

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); // not (1 << 3)
14
15     // Declaration: BUTTONSTATE sb_button_getState(BUTTON btn);
16     BUTTONSTATE state = sb_button_getState(BUTTON0); // not
        ↪ sb_button_getState(0)
17     /* ... */
18 }
```

- Use predefined types
- Only use explicit integer values if necessary

Bits & Bytes

Zahlensysteme

- Zahlen können in unterschiedlichen Basen dargestellt werden
  - $\Rightarrow$ 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

Number Systems

- Numbers can be represented using different bases
  - $\Rightarrow$ Usually: decimal (10), hexadecimal (16), octal (8) and binary (2)
- Nomenclature:
  - Bits: Digits of binary numbers
  - Nibbles: Groups of 4 bits
  - Bytes: Groups of 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

- Beispiel:

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

Bit Operations

- Bit operations: Bitwise logical expressions
- Possible operations:

~	
0	1
1	0

not

&	0	1
0	0	0
1	0	1

and

|	0	1
0	0	1
1	1	1

or

^	0	1
0	0	1
1	1	0

exclusive or

- Example:

$$\sim 1001_2 = 0110_2$$

$$1100_2 \;\&\; 1001_2 = 1000_2$$

$$1100_2 \;|\; 1001_2 = 1101_2$$

$$1100_2 \;\hat{}\; 1001_2 = 0101_2$$

Shiftoperationen

- 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 `1`er geshiftet.

Shift Operations

- Example:

```
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

- Setting single 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**

- **Caution:**
  When shifting `signed` variables, the behaviour of the `>>`-operator is not well defined in every case.

Assignment: 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

assignment: snake

- Snake consisting of adjecent LEDs
- Length (1 to 5 LEDs) is configured with the potentiometer (`POTI`)
- Speed depends on the environment brightness (`PHOTO`)
  - ⇝ The brighter the environment is, the faster the snake should move
- Mode of the snake can be toggled with a button (`BUTTON0`)
  - Normal: Switched on LEDs represent the snake
  - Inverted: Switched off LEDs represent the snake

⇒ You *should* work on the assignment in teams of two:
The `submit` scripts asks for your 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.

General Remarks

- Variables in functions behave similar to Java/Python
  - ⇝ To solve the assignment, only local variables are necessary
- The C compiler reads files from top to bottom
  - ⇝ Functions have to be declared in the right order:
    1. `wait()`
    2. `drawsnake()`
    3. `main()`

⇒ Details on compiler internals are discussed in the lecture.

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

Description of the Snake

- Position of its head
  - Number associated with a LED
  - Range of value $\{0, 1, \ldots, 7\}$
- Length of the snake
  - Integer in range of $\{1, 2, \ldots, 5\}$
- Mode of the snake
  - Normal or inverted
  - Can be represented as 0 and 1
- Speed of the snake
  - Here: Number of iterations of an active waiting loop

Zerlegung in Teilprobleme

- Basisablauf: Welche Schritte wiederholen sich immer wieder?
- Vermeidung von Codeduplikation:
  - ⇝ Wiederkehrende Teilprobleme in eigene Funktionen auslagern

Divide-and-conquer

- Basic program flow: Which steps do always repeat?
- Prevent duplicate code:
  - ⇝ Reoccurring problems can be addressed by helper functions

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 }
```

Basic Rundown snake

- Basic program flow: Represent snake, move snake, ...
- Pseudo code:

```
01 void main(void) {
02   while(1) {
03     // calculate length
04     length = ...
05
06     // draw snake
07     drawSnake(head, length, mode);
08
09     // put head to next position
10     ...
11
12     // wait and determine mode
13     ...
14
15   } // end of main loop
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

Representation of the Snake

- Parameters of representation
  - Position of the head
  - Length
  - Mode
- Function signature:
  ```
  void drawSnake(uint8_t head, uint8_t length,
  ↪ uint8_t modus)
  ```
- Representation depends on following parameters:
  - Normal mode (glowing snake):
    - Switch on all LEDs that belong to the snake
    - Switch off all remaining LEDs
  - Inverted mode (dark snake):
    - Switch off the LEDs belonging to the snake
    - Switch on all remaining 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

The Modulo Operator

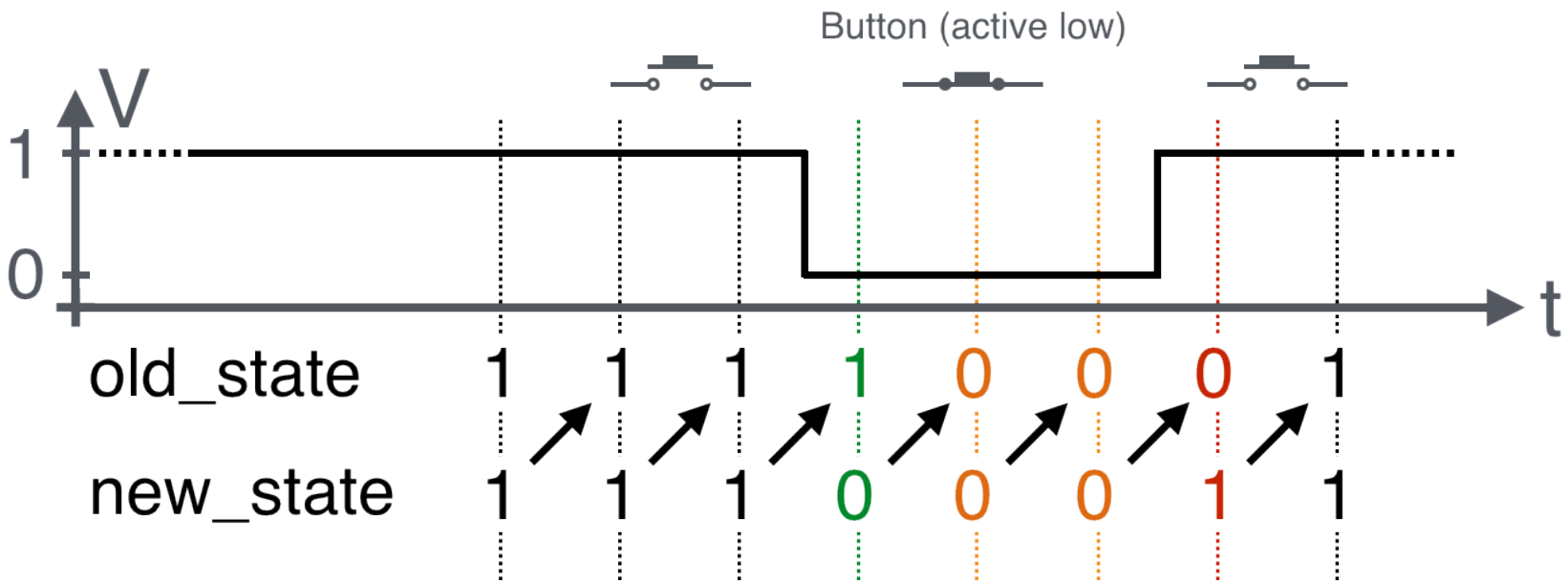
- Moving the snake
  - Modify the position of the head independent of the direction of movement
  - Problem: What happens at the end of the LED band?
- A solution: The modulo operator %
  - Remainder of an integer division
  - **Attention**: In C the result is negative for negative divisors
  - Example: `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**

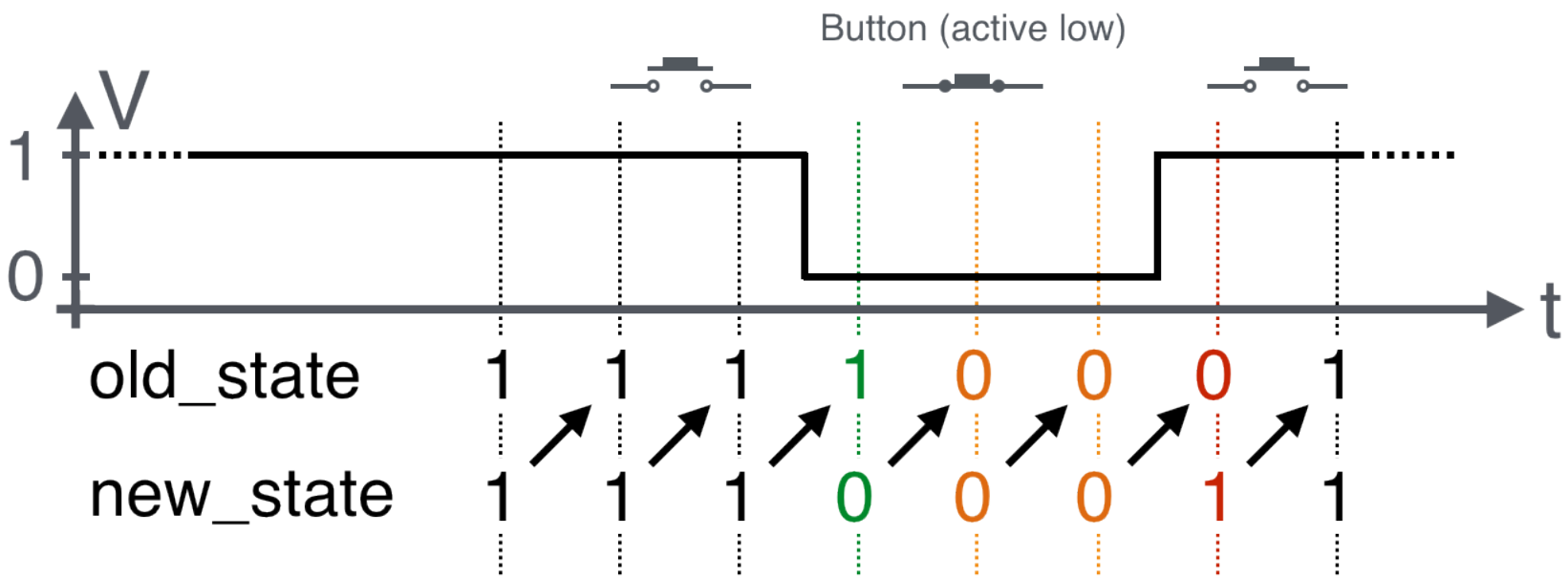
  → Nicht für Aufgabe relevant, es gibt PRESSED und RELEASED
  - Später: Realisierung durch Interrupts



Edge Detection without Interrupts

- Active waiting between movements of the snake
  - Detect whether the button has been pressed
  - Detect an edge by **cyclic polling** the level
  - Differentiate between **active-high** & **active-low**

  → Not relevant for implementation, use PRESSED and RELEASED
  - Later: Implementation using interrupts



Hands-on: Signallampe

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

Hands-on: Signal Lamp

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/ss25/spic/uebung/spicboard/libapi`
- Quelltext kommentieren

Hands-on: Signal Lamp

- Send Morse signals via `RED0`
- Controllable with `BUTTON1`
- Usage of library functions for button and LED
- Documentation of the library inside the SPiC IDE or via
  `https://sys.cs.fau.de/lehre/ss25/spic/uebung/spicboard/libapi`
- Insert comments in the source code