Variables
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 (SPI BOARD/ATMEGA328PB only has 2048 byte SRAM)
↝ Only use as little memory as necessary!

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); // not (1 << 3)

    // Declaration: BUTTONSTATE sb_button_getState(BUTTON btn);
    BUTTONSTATE state = sb_button_getState(BUTTON0); // not
    ↬ sb_button_getState(0)
    /* ... */
}
```

- Use predefined types
- Only use explicit integer values if necessary
Bits & Bytes

Number Systems

- Numbers can be represented using different bases
  - 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
Bit Operations

- Bit operations: Bitwise logical expressions
- Possible operations:

<table>
<thead>
<tr>
<th>~</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&amp;</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- not
- and
- or
- exclusive or

Example:

\[ \begin{align*}
\sim & \quad 1001_2 \\
& \quad 0110_2 \\
& \quad 1100_2 \\
& \quad 1000_2 \\
\wedge & \quad 1001_2 \\
& \quad 1000_2 \\
\mid & \quad 1001_2 \\
& \quad 1101_2 \\
^ & \quad 1001_2 \\
& \quad 0101_2 \\
\end{align*} \]
Shift Operations

- Example:

```c
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`
```

- Setting single bits:

```c
(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`
```

- Caution:

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

---

**assignment: snake**
Snake consisting of adjacent 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

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.
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

Divide-and-conquer

- Basic program flow: Which steps do always repeat?
- Prevent duplicate code:
  - Reoccurring problems can be addressed by helper functions
- External visibility: Scope should be as restricted as possible
  - Is the state only relevant for one function?
    - Local variable
  - Are more than one function accessing the same state?
    - Global/module local variable
Basic Rundown snake

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

```c
void main(void) {
    while(1) {
        // calculate length
        length = ...

        // draw snake
        drawSnake(head, length, mode);

        // put head to next position
        ...

        // wait and determine mode
        ...
    }
} // end of main loop
```

Representation of the Snake

- Parameters of representation
  - Position of the head
  - Length
  - Mode

- Function signature:
  ```c
  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
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 a integer division
  - **Attention**: In C the result is negative for negative divisors
  - Example: \( b = a \mod 4; \)

<table>
<thead>
<tr>
<th>a</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>-1</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

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**
  - Later: Implementation using interrupts
Hands-on: Signal Lamp

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

- 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/SS24/spic/uebung/spicboard/libapi
- Insert comments in the source code