SLP-assignment #4: game

(12 points, in groups of two)

Implement a skill game (file `spiel.c`) to train the hand-eye coordination. A cursor moves over the LED stripe of the SPiCboard. When `BUTTON0` is pressed, the current state of the LED pointed by the cursor will be switched: Goal of the game is to light up all 8 LEDs.

1. At the beginning of the game LED0 – LED7 are switched off.
2. The currently reached level, starting at level 1, has to be shown on the 7-segment display.
3. The cursor moves sequentially from LED0 to LED7 and back to LED0. To visualize this, the LED at the current position of the cursor is temporarily inverted (a switched *off* LED will be switched *on* and vice versa). Make sure that the cursor does not wait twice at the start and end of the LED stripe.
4. Pressing `BUTTON0` keeps this inversion. I.e., the temporary inversion is now permanent, even if the cursor moves on.
5. As soon as all LEDs are lit up, the level is cleared. Then, a victory sequence follows:
   (a) LED7 – LED0 are switched off one by one, beginning with LED7
   (b) The cursor moves from LED7 to LED0 and back.
   (c) The LEDs are switched on again starting with the outermost ones towards the center (starting at LED7 and LED0 simultaneously) and then are switched off from the center outwards.
6. The game continues with the next level (the cursor speed increases and approaches a maximum speed) and starts again. The speed should increase *more significantly* in the first few levels than in the last ones.

Your program should be divided into two main parts, that have to be called from the `main()` function: `play()` (game logic) and `show_win()` (victory sequence). Think about suitable return values and parameters for these functions. You can use additional auxiliary functions to encapsulate functionality.

`play()` The function `play()` should contain the implementation for one level. The speed for the level should be passed as a parameter upon calling. When the level is finished, the function should return.

`show_win()` The function `show_win()` shows the victory sequence. The individual steps of the sequence has to be made visible by short waiting periods in between.

Wherever your program waits, you have to wait **passively**! To do so, you can use the `libspicboard` (see remarks).

Hints:

- Use the `libspicboard` for addressing the 7-segment display (`sb_7seg_showNumber()`) as well as for passive waiting (`sb_timer_delay()`).
- Only use loops and bit operations to create bit masks for the LED stripe and then *only* use the function `sb_led_setMask()` to address the LEDs.
- Use local variables where ever possible and use global variables with suitable visibility only where required.
- The usage of the `button` module of the `libspicboard` is **not** allowed!
  - Directly configure the interrupt handling for `BUTTON0`. It is wired to pin `PD2` and therefore connected to the external interrupt source `INT0` of the ATmega.
  - Each button press is signaled by a falling edge.
  - Multiple presses during the same cursor position do not have to be taken into account.
  - The interrupt service routine has to be as short as possible.
- Always give a reason why you use the `volatile` keyword. If the same reasoning holds for multiple variables, you can justify them together.
- In the directory `/proj/i4spic/pub/aufgabe4/` you can find the file `spiel.elf` which serves as the reference implementation.

Deadline

T01	01.06.2026	18:00:00
SLP	03.06.2026	18:00:00

Autonomous Solving

All tasks must be completed independently. The generative use of AI tools of any kind is prohibited.