General Remarks for the SLP Computer Exercises

- You need a login for the computer-science CIP pool to take part in exercise course. If you do not have a login, one can be created via `https://account.cip.cs.fau.de`.
- Anyone, who registered for the exercises via Waffel, receives a project directory `/proj/i4spic/<login>/`, where `<login>` is a placeholder for your login name. A registration in the system is therefore mandatory to work on the assignments! The project directory is automatically integrated in the SPiC-IDE.
- The structure of the directory for the assignments has to be organized as follows:
  ```
  /proj/i4spic/<login>/aufgabe1
  /proj/i4spic/<login>/aufgabe2
  ...
  ```
- The assignments have to be submitted in the SPiC-IDE not later than the deadline.
  Alternatively, they can be submitted via
  ```
  /proj/i4spic/bin/submit aufgabeX
  ```
  (with `X` = `1 ... n`). This script copies the files required by the assignment description from the corresponding directory. Before the deadline, any program can be submitted an arbitrary number of times – the most recently submitted version will then be graded after the deadline.
- To check the last (and therefore valid) submission, the SPiC-IDE can be used or via
  ```
  /proj/i4spic/bin/show-submission aufgabeX
  ```
  you can view the last submitted program. To only view differences between the last submission and the current status in the project directory, the option `-d` can be added.
  ```
  /proj/i4spic/bin/show-submission -d aufgabeX
  ```
- The latest date for submission can be seen in the SPiC-IDE or with the call of:
  ```
  /proj/i4spic/bin/get-deadline aufgabeX
  ```
- Grading of a program submitted after the deadline can only be done in **well reasoned and exceptional cases**. You need to address the tutor directly who will then decide individually. An earlier submission before the deadline is *not* overwritten by a late submission. If in doubt, the first one is therefore graded.
- This term, the SPiCsim as well as the SPiCboard serves as a reference for the correction of the assignments. Please make sure that your solution behaves on earch of the platforms exactly as required by the assignment description.
- If not specified further, you need to use the same name for the C source file as the title of the assignment is called. I.e., if the assignment is called *blink*, the program should be created as `blink.c`.
- Further information can be found online:
  `https://sys.cs.fau.de/lehre/ss25/spic/`
- The documentation of the `libspicboard` can also be found there:
  `https://sys.cs.fau.de/lehre/ss25/spic/uebung/spicboard/libapi`

SLP-assignment #3.4: button

(14 points, no groups)

In this exercise, you will implement part of the `button` module, which is responsible for reading the push buttons in the `libspicboard`.

Strictly follow the interface defined in `button.h` and implement it in the file `button.c`. Additionally, write an application in the file `test-button.c` to test the functionality of your implemented `button` module.

Part a: Initialization (3 points)

Implement the module-internal initialization function `sb_button_init()`, which handles the hardware initialization.

Pins `PD2` (`BUTTON0`) and `PD3` (`BUTTON1`) must be configured as inputs and the internal pull-up resistors must be enabled. Ensure that the hardware initialization is performed only during the first call to any interface function

Ensure that all internal helper functions within the module have the correct visibility.

Part b: Deactivate Callback (2 points)

For simplicity, you do not need to implement callback functionality. However, the interface functions `sb_button_registerCallback()` and `sb_button_unregisterCallback()` should return `-1` to indicate that the functionality is missing.

Part c: Read Button (5 points)

Now implement the function `sb_button_getState()`, which returns the current state (`PRESSED`, `RELEASED`) of a button (`BUTTON0`, `BUTTON1`) to the caller. Note that both buttons are configured as `active-low`. The button reading should be done as follows:

- The button `BUTTON0` is hardware-debounced, so you only need to check whether pin `PD2` is at the corresponding level for `PRESSED` or `RELEASED`.
- For button `BUTTON1`, implement a simple debounce routine. Sample the level of pin `PD3` 99 times at intervals of $5\mu s$. Return the level that occurred most frequently.
- Use the function `_delay_us()` from the header `<util/delay.h>` to implement active waiting between the samples.
- Ensure that your implementation returns the correct error values for invalid parameters. Refer to the `libspicboard` documentation for details.

Part d: Testing the Module (4 points)

Now implement an application in the file `test-button.c` that tests all functions of the module from parts a) to c) (module test). The test should particularly have the following properties:

- Query each return value of the functions provided by the module at least once.
- If applicable, test different combinations of input parameters.
- To test the `sb_button_getState()` functionality, you may light up an LED in an infinite loop as long as `BUTTON0` or `BUTTON1` is pressed.

Make sure that successful and error cases can be clearly distinguished.

Hints:

- Since you do not use the parameters in the functions `sb_button_registerCallback()` and `sb_button_unregisterCallback()`, the compiler may emit a `-Wunused-parameter` warning. You may ignore this warning.
- The linker always uses the llocalffunctions defined in your own source files. Only if a function is not defined there, the libraries will be searched. To test with the button functions from `libspicboard` instead of your own implementation in `button.c`, simply comment out your implementation, for example by adding the line

  `#if 0`

  at the beginning of the file, and a line

  `#endif`

  at the end of the file. To enable your implementation again, change the `0` to a `1`.
- Always give a reason why you use the `volatile` keyword. If the same reasoning holds for multiple variables, you can justify them together.

Deadline

Use script in CIP pools: `/proj/i4spic/bin/get-deadline aufgabe3.4 Txx`