SLP-assignment #5: solar installation

(15 points, groups of two)

Design a control panel for an at home solar power plant in the file `solar.c`. In the initial state, the panel should be switched off and should not display any values. If a user wants to see the relative power generated at the moment, the panel can be switched on via Button0. This activates the 7 segment display and shows the relative power generation based on the PHOTO sensor. Then, a user can press Button1 to switch between modes and activate the LED mode, where the same relative performance is shown as a level on the LED strip. After a certain time has passed without the user pressing any button, the panel will switch itself off again. The task is divided into two parts. In the second part, an additional reset logic has to implemented such that a long press of Button1 will reset the panel while a short press still switches between the two previously described modes.

Part a) Basic Logic of the Control Panel (11 points)

Initially, the display of the panel is the 7 segment display of the SPiC Board. While in the LED mode, the LED mode is used via `sb_led_showLevel()` to show the relative generated power of the solar power plant. Pressing `Button0` while in the switched off state, the control panel activates itself in the 7 segment mode to show the generated power. To get a reading of the power, the photosensor `PHOTO` has to be used and the value has to be updated regularly after 1s. The read value from the photosensor has to be divided by the maximum possible value to generate a percentage between 0 and 99 that will fit on the 7 segment display. Pressing `Button1` will then switch the mode to the LED state (or back to the 7 segment mode). For the LED mode, the relative value of the photosensor should be interpreted as a level indicator. If 15s pass without a button pressed, the control panel will switch itself off again and wait for `Button0` to be pressed.

In detail, the controller should work as follows:

- The initial state is the off state and only `Button0` shoudl activate the panel.
- As `Button0` gets pressed, the control panel switches itself on and shows the string “So” on the 7 segment display for one second by using the function `sb_7seg_showString()`
- After this one second, the control panel will enter the switched on state in the 7 segment mode. Therefore, the value of the photosensor can be read with `sb_adc_read()` and then be mapped onto the interval from 0 to 99 to be representable on the 7 segment display. This value should then be updated every second.
- If `Button1` is not pressed for 15 seconds, the control pnale will switch itself off again.
- Pressing `Button1` in either of both switched on state will transition in the other switched on state: from 7 segment mode to LED mode and vice versa.
- In the LED mode, the value of the solar powerplant should also be read every second and then be shown on the LED strip with `sb_led_showLevel()`.

Part b) Logic for a Reset (4 points)

- In this part, the logic for performing a reset of the control panel should be implemented. To make this possible, we need to distinguish between short and long button activations for `Button1`.
- To make this possible in the grand scheme of finite state machines, we implement two transition states, one for switching from the 7 segment mode to the LED mode and one for the other way round.
- As soon as the button is pressed, we will enter the correct transition state. Then, the transition states can be exited in two different ways:
  - As soon as the activation of the button ends, the mode is switched, 7segment to LED or vice versa.
  - If one second passes and the button is still pressed, a reset is performed and the control panel will reenter the boot state where the string “So” is shown on the display for one second. Then, the operation continues as described above.

– It is vital to ignore all unnecessary interrupts and only have a selected group active at any time.

Make sure that the microcontroller enters a sleep mode whenever no calculations are required. This can be done using functions from `avr/sleep.h`.

Always remember the correct usage of the `volatile`-keyword. For each declaration of a `volatile` variable, add a comment explaining why the keyword is needed there.

Hints:

- Using a finite state machine can be very helpful. Try to think about which states and which transitions there are. If unsure, refer to the exercise slides.
- Use the modules `led` and `7seg` of the `libspicboard` for all outputs.
- However, do *not* use the `button` and `timer` module of the `libspicboard`!
  - Instead, you have to configure the interrupt handling and -handler for `BUTTON0` and `BUTTON1` directly. They are wired to pins `PD2` (`BUTTON0`) and `PD3` (`BUTTON1`) respectively and therefore connected to the external interrupt sources `INT0` and `INT1` of the ATmega. However, consider that the interrupt detection could be different for these two buttons.
  - For the timing, you should use `TIMER0`. You may use the overflow interruption `OVF` (errors up to 50ms can be tolerated). Choose the most resource efficient `prescaler`.
- Design your program in such way that `main()` implement only once the logic for entering a sleeping state. In particular, do not call or implement the sleep state for each change of state individually.
- Always give a reason why you use the `volatile` keyword. If the same reasoning holds for multiple variables, you can justify them together.
- Do not make any assumptions about the initial state of the hardware.
- In the directory `/proj/i4spic/pub/aufgabe5/`, you can find the file `solar.elf` which contains a reference implementation.

Deadline

Use script in CIP pools: `/proj/i4spic/bin/get-deadline aufgabe5 Txx`