Exercise Sheet 3 - Interfacing External ICs

Simple external peripherals have been controlled in the previous exercises. In practical use cases, however, it's all about interfacing more complex chips by using external buses. This even holds true for input and output devices, where LEDs and buttons are controlled by dedicated chips in order to reserve microcontroller pins for other, more important tasks. Therefore, we will drive the LEDs `D1` to `D4` and read the buttons `PB1` to `PB4` by using a so-called *shift register* as an intermediate device. While these devices tend to be replaced by more advanced circuitry nowadays (e.g. FPGAs), the signals you send to the shift register are very similar to the signals of the widely used SPI bus. Basically, you implement one of today's most important bus protocols - on your own!

Shift register:
You should already know the basic structure of a shift register from preceding lectures. In principle, it is a device to convert serial data streams into parallel ones and vice versa. You will need 5 out of 8 data lines of the shift register ICs. Specifically, these are:

- `CON2:P2.5 /CLR`: *If applying a 0 to this pin, all outputs of the shift register are reset to 0. Hence, you should set it to 1 for the execution of your program.*
- `CON2:P2.2 S0`, `CON2:P2.3 S1`: *These two pins control the mode of the first shift register.*
- `CON2:P2.0 S0`, `CON2:P2.1 S1`: *These two pins control the mode of the second shift register.*
- `CON2:P2.4 CK`: *Whenever* `CK` *changes from logic 0 to logic 1, the action which is determined by the given mode (see S0 and S1) is performed.*
- `CON2:P2.6 SR(2)`: *This pin specifies the next bit that is shifted into the second shift register.*
- `CON2:P2.7 QD(1)`: *With this pin, you can read out the fourth out QD of the first shift register.*

The possible modes and their effects can be found on page 2 of the data sheet (`74HC194_OK.pdf`*) of the shift register.*
The general principle of a single shift operation is as follows:
Select the corresponding mode, set `SR` *depending on your data and let the clock change once from 0 to 1. You might now read from* `QD` *and continue with the next action.*

An example to illustrate: Suppose you want to enable LED `D2`, while all other LEDs should be turned off. For simplicity, we assume that all pins have been correctly initialized. Listing 1 shows the code to perform the required operation.

Task 1

Realize a running light system (in German: *Lauflicht*, see footnote [1] for an illustration) with the LEDs D1 to D4. One LED of the running light system must always be illuminated, therefore there are 4 states in total. The buttons PB1 to PB4 realize the well known functions of a tape recorder to control the running light system as follows:

- The inital state should be pause and all LEDs are initialized and turned off. (**1 pt.**)
- PB3 is the **playback button**. If it is pressed, the sequence should continue from the current position (**1 pt.**) and run with approximately 1 pass (4 state changes) per second from D1 to D4. Playback continues when the button is released. (**1 pt.**)
- PB2 should be the **pause button**. If it is pressed, the running light should stop at the current position and stay paused when the button is released. (**1 pt.**)
- PB4 is the **fast-forward button**. As long as PB4 is pressed, a fast light with about 2 complete passes (8 state changes) per second should run from D1 to D4. (**1 pt.**)
  After PB4 is released, it should return to its previous state (pause or normal playback). (**1 pt.**)
- PB1 is the **rewind button**. As long as PB1 is pressed, a fast light sequence with about 2 complete passes (8 state changes) per second should run from D4 to D1. (**1 pt.**)
  After PB1 is released, it should return to its previous state (pause or normal playback). (**1 pt.**)
- Make sure that your program responds to the input regardless of the LED's state. There is no need to consider simultaneous pressing multiple buttons. (**1 pt.**)

Note:
Running lights can't jump over intermediate states. Therefore, the LED can only switch to adjacent LED states. The allowed states are defined as follows:

State 4	*State 1*	*State 2*	*State 3*	*State 4*	*State 1*
...	●●●●	●●●●	●●●●	●●●●	...

Bonus:
Use the LEDs D5, D6 and D7 as a rotating running light, operating in parallel to the linear light. All features should be synchronous on both scales. (**1 pt.**)

Hint:
This time, please also consider the **PxSEL** register for initialization. For example, CON2:P2.6 and CON2:P2.7 are no regular pins by default, as they are input and output pins of a low-frequency oscillator. Therefore, you have to set them to the normal pin behavior before starting with the exercise. This is done by clearing the appropriate bits of the registers P2SEL and P2SEL2.

[1] http://de.wikipedia.org/wiki/Lauflicht

Task 2

Create a file `feedback.txt` with a brief feedback statement, which contains specific problems and issues you experienced while solving the exercise, additional requests, positive remarks and alike. Import this text file `feedback.txt` in your **C**ode **C**omposer **S**tudio (CCS) project, so that you can upload it together with your software deliverable. (**1 pt.**)

Listing 1: Example for controlling the shift register

```
// SR 1 is responsible for buttons and ignored in the following.

// Set pin direction
P2DIR |= (BIT0 | BIT1 | BIT4 | BIT5 | BIT6);
// Modify XTAL pins to be I/O
P2SEL &= ~(BIT6 | BIT7);

// Clear / reset the shift register , i.e. turn all LEDs off.
P2OUT &= ~BIT5;
// stop the reset mode (/CLR = 1) and
P2OUT |= BIT5;

// Reset the clock signal to 0
P2OUT &= ~BIT4;

// Set the shift register 2 mode right shift mode (S0 = 1, S1 = 0) ,
P2OUT |= BIT0;
P2OUT &= ~BIT1;

// apply the data to be shifted at SR (SR = 1 to insert a 1).
P2OUT |= BIT6;

// apply a rising clock edge => shift data in
P2OUT |= BIT4;
// reset the clock
P2OUT &= ~BIT4;

// set SR to 0, because we don't want D1 to be on
P2OUT &= ~BIT6;

// apply a rising clock edge => shift data in
P2OUT |= BIT4;
// reset the clock
P2OUT &= ~BIT4;

// => D2 is enabled , D1 isn't
```