

Exercise Sheet 1 - Digital I/O and Interrupts¹

In this exercise, pins should be used as inputs and outputs. On the circuit board, you can find two buttons **PB5** and **PB6**. They can be connected to the microcontroller via the header pins **PB5 (X11)** und **PB6 (X12)**. When you press these buttons, a physical connection to the ground potential is established. Once you configured the registers **PxDIR** and **PxSEL**, you can query the pin's logical state by reading the 'input register' (**PxIN**²). Please consider the wiring of the buttons in order to determine if a pullup-/pulldown resistor is needed. The 'Pullup/Pulldown Resistor Enable Register' (**PxREN**³) will allow you to internally connect pullup-/pulldown resistors to the I/O-ports.

Note:

Unless noted otherwise, you should always merge all tasks of an exercise sheet into one piece of source code. The required tasks are intended to run together.

Warning: Non-compliance to 'Note:' sections can lead to point deduction.
Please keep that in mind for all exercise sheets!

Task 1

- a) Connect the button **PB5** to **CON3:P1.3** and the button **PB6** to **CON3:P1.4**. Then, connect the red LED **D6** to **CON3:P1.5**, the green LED **D5** to **CON3:P1.6** and the blue LED **D7** to **CON3:P1.0**. Lastly, connect the yellow LED **D9**, which is located to right of jumper **JP3**, by connecting **CON3:P1.7** to the right pin of **JP3**.

Write a program to monitor the buttons **PB5** and **PB6**. When entering a state, also when changing from one state to another, the corresponding description applies. There are four possible states with respect to the pushbuttons which are either pressed (1) or released (0):

Table 1: State specification

State	PB5	PB6	D5	D6	D7	D9 (=!D6)
1	0	0	OFF	OFF	OFF	ON
2	1	0	OFF	ON for 250 ms, then OFF	OFF	OFF for 250 ms, then ON
3	0	1	ON	OFF	OFF	ON
4	1	1	OFF	OFF	ON	ON

Implement a program to follow the behavior of table 1. Use **polling** to monitor the buttons, therefore actively read input pin voltages instead of using an **Interrupt Service Routine (ISR)**. In order to prevent accidental changes from one state to another, make sure to debounce the buttons in your program. **(1.5 pts. per state)**

¹<https://en.wikipedia.org/wiki/Interrupt>

²see MSP430x2xx Family User's Guide: Section 8.2.1

³see MSP430x2xx Family User's Guide: Section 8.2.4

- b) Export the code which controls the red LED **D6** and the yellow LED **D9** into a separate function. An example of how to use functions is shown in the C cheat sheet. (1 pt.)
- c) Now **modify** the readout method for **PB5**. Therefore, use an interrupt instead of polling (see listing 3). Make sure to maintain previous functionality and continue to poll **PB6**. Don't delete the polling code for button **PB5**, use `#define` to activate or deactivate parts of your code as described in listings 1 and 2. (2 pts)

Listing 1: 'polling' defined.

```
#define polling

#ifdef polling
// Will be compiled
#endif
#ifndef polling
// Won't be compiled
#endif
```

Listing 2: 'polling' not defined.

```
//#define polling

#ifdef polling
// Won't be compiled
#endif
#ifndef polling
// Will be compiled
#endif
```

Listing 3: Example of the initialization and use of interrupts.

```
// For proper execution, please make sure that
// interrupts are globally enabled!

// Initialization
P1DIR &= ~BIT0; // Set as input
P1REN |= BIT0; // Enable pull-resistors
P1OUT |= BIT0; // Set to pull-up
P1IE |= BIT0; // Enable interrupt
P1IES |= BIT0; // High/Low-Edge
P1IFG &= ~BIT0; // Clear interrupt flag

// ...other code

// Port 1 interrupt vector
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void) {
    // Do something (but keep in mind you're still in the interrupt,
    // so don't let it take TOO long).
    // Also note that all variables that you change within this function
    // must be declared 'volatile'.
    // Clear interrupt flag (here - as an example - the flag of P1.0).
    P1IFG &= ~BIT0;
}
```

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 **Code Composer Studio (CCS)** project, so that you can upload it together with your software deliverable. (1 pt.)

Note:

1. Please **always** name both your project in CCS as well as the ZIP file you upload according to the following structure, replacing the expressions within the brackets:

Exercise_[ExerciseNo]_[YourLastName]

So, if you upload your solution for exercise 3 and your name is John Doe, then you have to upload the project Exercise_3_Doe within the ZIP folder Exercise_3_Doe.zip

2. Please **always** include your name in both your main.c (in a header comment) and your feedback.txt.
3. To export your project, please **always** use: File > Export... > General > Archive File, then select the required project and type the name of the ZIP file in the field 'To archive file:'.