Exercise sheet 5 - Pulse Width Modulation

Pulse Width Modulation (PWM):
Pulse width modulation is a concept in which a rectangular signal with a constant frequency and variable duty cycle (remember: ratio of on-time to period time) is generated by alternating in-between logical high and low levels with a certain delay time. Moreover, a logic level PWM signal can be converted into an analog signal using a low-pass filter (PWM + Low-Pass is a simple digital-to-analog converter for very low sampling frequencies). In this exercise, we're going to use the microcontroller's PWM generator as a flexible frequency generator, keeping the duty cycle at 50 %.

Listing 1: Example of how to create a PWM signal.

```
P3DIR  |= BIT6;            // P3.6 output
P3SEL  |= BIT6;            // P3.6 TA0.2 option
TA0CCTL2 = OUTMOD_3;       // CCR2 set/reset
TA0CCR0  = 1000;           // PWM Period: 1000 us
TA0CCR2  = 500;            // CCR2 PWM duty cycle (50 %)
TA0CTL = TASSEL_2 + MC_1;  // SMCLK; MC_1 -> up mode;
// MC_2 -> cont mode,
// MC_3 -> up-down-mode
// In continous mode, internal reference
// counts up to 0xFFFF before starting at
// 0x0000 again. In any other mode it counts
// up until it hits TACCR0
// (p. 364 in the family guide)
```

Listing 2: Example for using arrays.

```
int data[6] = {440, 440, 440, 349, 523, 440};
int counter;
for (counter = 0; counter < 6; counter++) {
  playNote(data[counter]);
}
```

Task 1

Connect the Buzzer `X1:Buzzer` with `P3.6`. Also connect the button `PB5` with `P1.3` and button `PB6` with `P1.4`. Set the Jumper `JP5` to `VFO`.

a) Implement a jukebox which can replay two melodies of your choice by using the piezo buzzer. To generate the audio signal, use the PWM functions of the microcontroller to apply certain frequencies at certain time points. Store the melodies in an array to keep your code short (**1 pt. PWM + 1 pt. per melody + 1 pt. storing in array**). Configure your program that both melodies will be replayed after each other, with a pause of about one second in-between the sounds.

b) Now, extend your program so that no melody is played by default (comment out the old play section from task 1a). Instead, capture PB5 using interrupts (**1 pt.**) and implement the following selection method:

- If PB5 is pressed once within one second, play melody 1 (**0,5 pt.**)
- If PB5 is pressed twice in the same time interval, play melody 2 (**0,5 pt.**)
- Turn off the interrupt while playing a melody.

☞ **Reminder:** An example of an interrupt service routine was given on exercise sheet 2.

c) A piezo element can alternatively be used as a vibrational sensor. Extend your program so that it also responds to a knocking signal on the piezo (carefully knock your board on the table for activation) and not only to `PB5`. For this purpose, do not connect the piezo to another pin, but use `P3IN` to read it. (**2 pt.**).

☞ **Note:** It may be necessary to activate the **pull-up resistor** or the **pull-down resistor** of the piezo pin with the register textbf`PxREN` to dissipate the charge being created at the piezo's surface.

d) Implement button `PB6` as a pause / resume button (**1 pt.**).

Task 2

a) 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, etc. (**1 pt.**).

b) Import this text file `feedback.txt` in your Code Composer Studio project, so that you can upload it together with your software deliverable.