Practical Exercise – Mikrocomputertechnik
Prof. Dr. habil Stefan J. Rupitsch
Laboratory for Electrical Instrumentation
and Embedded Systems

Exercise Sheet 3 - Pulse Width Modulation

Pulse Width Modulation (PWM[1]):
*Pulse width modulation is a concept in which a rectangular signal with a constant frequency and variable duty cycle is generated by alternating in-between logical high and low levels with a certain delay time. A simple '**D**igital-To-**A**nalog **C**onverter' (DAC) for very low sampling frequencies can be implemented by applying a low-pass filter to a PWM signal.*

Note:
In this exercise, we're going to use the microcontroller's PWM capabilities as a flexible frequency generator. Therefore, the duty cycle is constant and the frequency varies in order to generate different sounds. The constant duty cycle in your program should be 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)
```

TASSEL_2 selects the source for the timer clock as SMCLK (Sub-Main Clock).
MC_1 configures the timer in "up mode," where it counts up to the value in TA0CCR0 and then resets.

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]);
}
```

[1] https://www.arduino.cc/en/Tutorial/PWM

Task 1

a) Connect the buzzer `BUZZER` to `CON3:P3.6`, button `PB5` to `CON3:P1.3`, button `PB6` to `CON3:P1.4` and set the jumper `JP5` to `VFO`. Implement a jukebox which can play two melodies of your choice with the piezo buzzer. To generate the audio signal, use the PWM functions of the microcontroller, so that you can apply certain frequencies at certain time points. Store the melodies in an array to keep your code short. Configure your program so that both melodies are played after each other, with a pause of about one second in-between the melodies. (**1 pt. PWM + 1 pt. per melody + 1 pt. storing in array**)

b) **Modify** your program so that no melody is played by default. Instead, capture `PB5` using interrupts (**1 pt.**) and implement the following selection method:

- If `PB5` is pressed once within one second, play melody one. (**0.5 pt.**)
- If `PB5` is pressed twice within one second, play melody two. (**0.5 pt.**)

Note:
*Make sure to turn off the interrupt for `PB5` while playing a melody. Use #define to activate or deactivate parts of your code, as described in **Exercise Sheet 1**, where you can also find an example of an interrupt service routine. There is no need to consider pressing the button more than twice.*

c) A piezo element can alternatively be used as a vibrational sensor. **Extend** your program from **Task 1 b)** so that it **also** responds to knocking signals in the same way as `PB5`. Carefully knock your board on the table once to play melody one, or knock twice for melody two. For this purpose, do not connect the piezo to another pin, instead use **P3IN** for readout. (**2 pts.**)

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

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

Note:
Controls via `PB5` should still be possible in pause mode. Therefore, in pause mode, you can either resume playback with `PB6` or start playback of either melody one or two by pressing `PB5` or knocking according to the tasks above.

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.**)