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

Exercise Sheet 5 - Watchdog Timers & Timer Modules

In this exercise, we will see two methods to recover from getting stuck under certain conditions.

1. **Watchdog**
   The Watchdog is a hardware timer that prevents or detects failures in microcontroller systems. During regular operation, the software must periodically tell the Watchdog timer that it's still alive and working properly and for example reset the Watchdog timer. When there is some non-intended error within the program, the Watchdog will not be reset and its counter will overflow. In such cases, actions to restore the functionality of the system, for example by restarting, can be carried out.
2. **Dynamic Control Systems**
   In case of errors, such as failure of the CLK system, program bugs or partial failure of the supply voltage, output pins of a microcontroller can be stuck at any static state (high, low or intermediate values). External circuits connected to these pins, such as a relay, can therefore be driven in non-intended states. This can be partially prevented by dynamic control principles, such as a PWM driven relais with a dedicated AC/DC circuit.

Task 1

Connect the button `PB5` to `CON3:P1.3` and the green LED `D5` to `CON3:P1.0`. Let the green LED `D5` blink with a frequenzy of 4 Hz. If the button `PB5` is pressed, the program should enter an endless loop, in which no further code will be executed. This results in an **artificial deadlock**. The Watchdog should recognize the inactivity of the microcontroller caused by the artificial deadlock and reset the microcontroller after about 5.5 seconds. (**2 pts.**)

Note:
The Watchdog timer must be initialized at startup with the desired time delay before overflowing. During regular operation the watchdog timer counts up but is regularly reset to zero and therefore doesn't overflow. However, when entering the artificial deadlock the timer **can not** be reset by your program and therefore the watchdog counter will overflow.
Consequently, you should not start the watchdog timer when entering the artificial deadlock!

To initialize the watchdog timer+, first, route the *Internal **V**ery-Low-Power **L**ow-Frequency **O**scillator **Cl**ock* (VLOCLK) to the ***A**uxiliary **Cl**ock* (ACLK) line. Modify the multiplexer with **LFXT1Sx** accordingly. See Figure 5-1 on page 277 in the *MSP430x2xx Family User's Guide*.
Afterwards, use the ACLK as a clock source for the watchdog timer+, which should be used in watchdog mode, as this allows to restart the MSP430 via ***P**ower **U**p **C**lear* (PUC). See Figure 2-1 on page 30.
Then, initialize the ***W**atchd**o**g **T**imer+ **Co**ntr**o**l* (WDTCTL) register. See page 363. As the password must be used, please note the assignment used in the template similarly to the following command:

```
WDTCTL = WDTPW | WDTSSEL | ...;
```

Task 2

a) Connect X3 to X10 and activate the heating resistor via JP4. Also, connect X4 to CON3:P3.5, X5 to CON3:P3.4 and the red LED D6 to CON3:P3.2. Then, connect the thermistor U_NTC to CON3:P1.5. **Modify** your code from **Task 1** in such a way that it resets the microcontroller after about 21.9 seconds. Again, please use #define to enable/disable parts of your code, in this case for the two different initializations of the watchdog timer+. (**1 pt.**)

b) The NTC's resistance will change with temperature. Your task is to implement a thermometer using the NTC for measurement and the LEDs D1 to D4 as well as the red LED D6 to display the temperature.
First, determine the range of possible output voltages of the NTC voltage divider with the ADC for the complete temperature range you can apply with the heater. Therefore, activate the heating resistor R28 with the static input of the relay REL_STAT and capture the maximum and minimum values of the ADC for the complete heating period.
Then, divide the value range into five equal subranges, which are represented by the LEDs D1 to D4 for the four lower ranges and the red LED D6 for the fifth range, where the red LED D6 indicates that the temperature is too high. (**1 pt.**)

Note:
The following code contains crucial information on how to configure the ADC. However, you will still need to adapt the code to determine the output voltages of the NTC.

Listing 1: Example Program for the use of the ADC.

```
/*
 * This programm will print the analog value at P1.7 to the serial
    console.
 *
 * Connect the following pins: P1.7 (CON3) to U_POT (X6)
 */

#include <templateEMP.h>
void main(void) {
  initMSP();
  // Turn ADC on; use 16 clocks as sample & hold time (see p. 559 in
     the user guide)
  ADC10CTL0 = ADC10ON + ADC10SHT_2;
  // Enable P1.7 as AD input
  ADC10AE0 |= BIT7;
  // Select channel 7 as input for the following conversion(s)
  ADC10CTL1 = INCH_7;

  while (1)
  {
    // Start conversion
    ADC10CTL0 |= ENC + ADC10SC;
    // Wait until result is ready
    while(ADC10CTL1 & ADC10BUSY);
    // If result is ready, copy it to m1
    int m1 = ADC10MEM;
    // Print m1 to the serial console
    serialPrintInt(m1);
  }
}
```

c) The thermometer's visual LED output shall be refreshed every two seconds. This shall be accomplished by configuring a timer (use timer 0, it is a timer of *type A*) to trigger an interrupt every two seconds. Read and display the temperature within the interrupt service routine. (**2 pts.**)

Note:
To use the timer, consider the timer registers *TACTL* (in CCS, use `TACTL`), TACCTLx (in CCS: `CCTL0`) and TACCRx (in CCS: `CCR0`) described in the Family User Guide p. 369ff. The interrupt service routine is given as follows:

Listing 2: Timer Interrupt Service Routine to be used.

```
// Timer A0 interrupt service routine
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A(void) {
   // your interrupt code goes here
}
```

d) **Extend** the main routine of your program with a simple controller, which always ensures that the temperature stays within the temperature ranges three and four. When the button `PB5` is pressed while the heater **is on**, you should end up in the overheating (fifth) range, due to the artificial deadlock. However, if button `PB5` is pressed while the heater is off, the system will be 'undercooled'. Oh no! But, not to worry! The watchdog will reset the system for you, after the given delay, before any damage occurs. (**1 pt.**)

e) However, with the given circuit board, dynamic control systems can be used as a more responsive method for preventing the NTC from overheating. Therefore, **modify** your controller from **Task 2d)**, so that it uses `X4` instead of `X5` to control the heater. Consider the schematic to see what has to be changed in the heating algorithm. Again, use #define to enable/disable parts of your code. Further, the temperature display from **Task 2c)**, using the interrupt, should still run in the background without any modification. Test if your program avoids overheating when jumping into an artificial deadlock by pressing button `PB5`. (**2 pts.**)

Task 3

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.**)