FALL 2025 EEL4742 – *Embedded Systems* - Quiz 2 Solution - Time: 25 Minutes

1. (4 points) Complete the code so that:
   (a) (2 points) the LED would toggle every 0.5 seconds.
   (b) (2 points) The LED would toggle every 32 seconds (no additional for-loop can be used).

```
#include <msp430fr6989.h>
#define RED_LED BIT0

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= RED_LED;
    P1OUT &= ~RED_LED;

    ____________;
    ____________;

    for(;;) {
        while((TA0CTL & _______) == 0) {}
        P1OUT ^= RED_LED;
        ____________;
    }}
```

Answer (part a): ACLK = 32,768 Hz → 0.5 s = 16,384 counts. So use up mode with CCR0 = 16384 – 1.

```
// Filled line:
TA0CCR0 = 16384;                // 0.5s
TA0CTL = TASSEL_1 | MC_1 | TACLR;
for(;;) {
        while((TA0CTL & TAIFG) == 0) {}
        P1OUT ^= RED_LED;
        TA0CTL &= ~TAIFG;
    }}
```

Answer (part b): ACLK = 32,768 Hz → 2 s = 64,536 counts. We can use divider (x8), and up-down (x2) to make it 16x (32 s).

```
// Filled line:
TA0CCR0 = 65535;                // 2s
TA0CTL = TASSEL_1 | ID_3 | MC_3 | TACLR;
for(;;) {
        while((TA0CTL & TAIFG) == 0) {}
        P1OUT ^= RED_LED;
        TA0CTL &= ~TAIFG;
    }}
```

2. (3 points) Assuming that ACLK = 32,768 Hz, we need an interrupt every 33 ms in up mode. Write the line of code to initiate the timer for this application. Then, calculate the accuracy of timer.

Answer: Here is the calculation for TA0CCR0.

$$\text{counts} = f_{\text{ACLK}} \times T = 32768 \times 0.033 \approx 1081.34 \rightarrow \text{TA0CCR0} = 1081$$

```
TA0CCR0 = 1081;
TA0CTL  = TASSEL_1 | MC_1 | TACLR | TAIE;
```

Actual timer period is as follows. So, accuracy (error) is 0.06%:

$$T_{\text{actual}} = \frac{1082}{32768} \approx 0.03302\,\text{s} = 33.02\,\text{ms}$$

$$\text{Relative error} = \frac{0.02}{33.00} \times 100\% \approx 0.06\%$$

3. (3 points) Complete the following code in a way that interrupt is used to toggle the LED every 2 seconds.

```
#include <msp430fr6989.h>
#define RED_LED BIT0

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= RED_LED;
    P1OUT &= ~RED_LED;

    TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR | TAIE;
    ____________;
    ____________;

    for(;;) {}
}

#pragma vector = _______________
__interrupt void TA0_ISR(void) {
    _______________;
    _______________;
    }
```

Answer: We need to configure the interrupt as follows:

```
// for the missing part in main()
TA0CTL &= ~TAIFG;
__enable_interrupts();

#pragma vector = TIMER0_A1_VECTOR
__interrupt void TA0_ISR(void) {
    TA0CTL &= ~TAIFG;
    P1OUT ^= RED_LED;
    }
```

4. (5 points) **Bonus** For a button-driven LED system, write the FSM (enum and its case switch) with the following rules:

- rotating between on, off, blinking per every button pressed.
- If no button press, stay in the current state (no resets).
- Entering ”blinking” or ”off” should reset a counter value.
- Entering ”on”, should turn on LED and increment counter value.

Answer: Here is the FSM constructs for it:

```
typedef enum { S_OFF, S_ON, S_BLINK } led_state_t;
static led_state_t state = S_OFF;
static unsigned int counter = 0;

// Call on each scan/tick with debounced button input.
void fsm_step(bool button_pressed) {
    if (!button_pressed) return;              // no press, then stay in current state

    switch (state) {
    case S_OFF:
        state = S_ON;                          // rotate OFF, then ON
        P1OUT |= BIT0;                         // entering "on": turn LED on
        counter++;                             // entering "on": increment counter
        break;

    case S_ON:
        state = S_BLINK;                       // rotate ON, then BLINK
        counter = 0;                           // entering "blinking": reset counter
        /* start blinking timer here if needed */
        break;

    case S_BLINK:
        state = S_OFF;                         // rotate BLINK, then OFF
        counter = 0;                           // entering "off": reset counter
        P1OUT &= ~BIT0;                        // ensure LED off
        break;
    }
}
```