

A1

(Part a) Timer_A is using a 100 KHz (100,000 Hz) clock signal. The timer's period in continuous mode for each ID (Input Divider) value:

- ID = 0 (No division):

$$\text{Period} = \frac{65536}{100,000} = 0.65536 \text{ seconds}$$

- ID = 1 (Divide by 2):

$$\text{Period} = \frac{65536}{100,000/2} = 1.31072 \text{ seconds}$$

- ID = 2 (Divide by 4):

$$\text{Period} = \frac{65536}{100,000/4} = 2.62144 \text{ seconds}$$

- ID = 3 (Divide by 8):

$$\text{Period} = \frac{65536}{100,000/8} = 5.24288 \text{ seconds}$$

(Part b) We aim for a 0.1 second timer period in up mode. Using the formula:

$$\text{TACCR0} = \frac{0.1 \times 100,000}{\text{ID divisor}} = (10,000 \div \text{ID divisor}) - 1$$

Possible solutions:

- ID = 0 (No division): TACCR0 = 10,000 - 1
- ID = 1 (Divide by 2): TACCR0 = 5,000 - 1
- ID = 2 (Divide by 4): TACCR0 = 2,500 - 1
- ID = 3 (Divide by 8): TACCR0 = 1,250 - 1

A2

(Part a) Using ACLK (32,768 Hz) in continuous mode, the period for each ID (Input Divider) is:

- ID = 0 (No division):

$$\text{Period} = \frac{65536}{32,768} = 2 \text{ seconds} \quad \text{or} \quad \frac{2}{65,536} = 30.52 \mu\text{s}$$

- ID = 1 (Divide by 2):

$$\text{Period} = \frac{65536}{32,768/2} = 4 \text{ seconds} \quad \text{or} \quad \frac{4}{65,536} = 61.04 \mu\text{s}$$

- ID = 2 (Divide by 4):

$$\text{Period} = \frac{65536}{32,768/4} = 8 \text{ seconds} \quad \text{or} \quad \frac{8}{65,536} = 122.07 \mu\text{s}$$

- ID = 3 (Divide by 8):

$$\text{Period} = \frac{65536}{32,768/8} = 16 \text{ seconds} \quad \text{or} \quad \frac{16}{65,536} = 244.14 \mu\text{s}$$

(Part b) Using a 5 MHz clock signal, we want a delay of 0.5 seconds. The maximum period achievable with no divider is:

$$\text{Max Period} = \frac{65536}{5,000,000} = 0.0131072 \text{ seconds}$$

Since the maximum period is far less than 0.5 seconds, it is not possible to directly configure Timer_A to achieve a 0.5-second delay without using additional techniques (e.g., multiple overflows, software counters).

A3

(Part a) In C, an `int` typically represents 16 bits on many embedded systems, such as the MSP430.

(Part b) No, increasing the upper bound to 90,000 would not necessarily double the delay. This is because the loop counter may overflow depending on the maximum size of the integer, which could lead to undefined behavior.

(Part c) Possible solutions to double the delay include:

- Use a `long int` or another larger data type to increase the range of the counter.
 - Repeat the loop multiple times, effectively increasing the number of iterations while avoiding integer overflow.
 - Use a hardware timer (like Timer_A) instead of relying on software loops.
-

A4

(Part a)

- CCIFG interrupt period:

$$\text{Period}_{\text{CCR}0} = \frac{500}{\frac{32,768}{8}} = 0.12207 \text{ seconds}$$

- TAIFG interrupt period:

$$\text{Period}_{\text{TAIFG}} = \frac{65536}{\frac{32,768}{8}} = 16 \text{ seconds}$$

(Part b) The TAIFG interrupt will be serviced first because it has a higher priority (priority level 2) compared to CCIFG (priority level 1).

(Part c) Changing the clock divider to 32 will result in:

- CCIFG interrupt period:

$$\text{Period}_{\text{CCR}0} = \frac{500}{\frac{32,768}{32}} = 0.488281 \text{ seconds}$$

- TAIFG interrupt period:

$$\text{Period}_{\text{TAIFG}} = \frac{65536}{\frac{32,768}{32}} = 64 \text{ seconds}$$

A5

(Part a) In LPM3, SMCLK is disabled. Since the clock driving of Timer_A is disabled, none of the interrupts will occur. If LPM3 was disabled, then TAR reaches CCR1, the CCR1 interrupt will trigger. After this, Timer_A continues to count, and once TAR overflows (reaches 65,535), the TAIFG overflow interrupt will trigger.

(Part b) To ensure the CPU wakes up to handle both the CCR1 and TAIFG interrupts, you must configure the system to exit low-power mode (LPM3) on both interrupts by clearing the LPM3 bits in the interrupt service routines.

(Part c) The CCR1 interrupt will occur when TAR reaches 30,000:

$$\text{Time}_{\text{CCR}1} = \frac{30,000}{1,000,000} = 0.03 \text{ seconds}$$