Quiz 4 Study Guide

1) **Given the configuration code of a timer, know how to calculate its interval. (Know which macro refers to which clock source or counting mode).**

```
void main(void)
{
    WDTCTL = WDT_MDLY_32;           // 32ms interval (default)
    P1DIR |= BIT0;                  // Set P1.0 to output direction
    SFRIE1 |= WDTIE;                // Enable WDT interrupt

    _BIS_SR(LPM0_bits + GIE);       // Enter LPM0 with interrupt
}

// Watchdog Timer interrupt service routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    static int i = 0;
    i++;
    if (i == 32) {                  // 31.25 * 32 ms = 1s
        P1OUT ^= BIT0;              // Toggle P1.0 using exclusive-OR
                                    // 1s on, 1s off; period = 2s, f = 1/2s = 0.5
        i = 0;
    }
}
```

2) **What are some of the properties of timers? What can they be used for? How is their power consumption? How are they affected by low power modes? How independent they are from the CPU?**

- Timers are simple digital counters that in active mode increment or decrement their value at a specified clock frequency.
- Initialization – Specify timer's operation mode, clock frequency, and whether it will raise an interrupt once the counter reaches a value.
- Power consumption – Using timers allows you to use power on the timer instead of counting through the CPU and leaving it on. Counting on the CPU is power hungry and cumbersome, use timers instead.
- Low power mode – _BIS_SR(LPM0_bits + GIE); // Enter Low Power Mode 0
- _BIS_SR(LPM3_bits + GIE); // Enter Low Power Mode 3
- CPU Independence – a different clock signal than the processor, it is possible either to turn off the processor or to work on other computations while the timer is counting.

SCG1[1]	SCG0	OSCOFF[1]	CPUOFF[1]	Mode	CPU and Clocks Status[2]
0	0	0	0	Active	CPU, MCLK are active. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	1	0	1	LPM1	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is disabled.
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5[3]	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this n RTC operation is possible when configured properly. See the *RTC* module for details.
1	1	1	1	LPM4.5[3]	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this n clock sources are disabled; that is, no RTC operation is possible.

(1) This bit is automatically reset when exiting low power modes. Refer to Section 1.4.1 for details.
(2) The low-power modes and, hence, the system clocks can be affected by the clock request system. See the UCS chapter for deta
(3) LPM3.5 and LPM4.5 modes are not available on all devices. See the device-specific data sheet for availability.

3) **Given a code for two timer channels, know how to calculate the duty cycle (The diagram representing different output modes will be provided during the quiz)**

https://www.ti.com/lit/an/slaa513a/slaa513a.pdf?ts=1603998875530&ref_url=https%253A%252F%252Fwww.google.com%252F

4) **Given the block diagram and the configuration code, be able to determine the source of a timer.**

variables, put in ratio.
↳ give variables

- **ACLK**: Auxiliary clock. The signal is sourced from LFXT1CLK with a divider of 1, 2, 4, or 8. (The calibration program for the serial link sets the divider to 4, but after the calibration it can be changed to any other values.) ACLK can be used as the clock signal for Timer A and Timer B.
- **MCLK**: Master clock. The signal can be sourced from LFXT1CLK XT2CLK (if available), or DCOCLK with a divider of 1, 2, 4, or 8. MCLK is used by the CPU and system.
- **SMCLK**: Sub-main clock. The signal is sourced from either XT2CLK (if available), or DCOCLK with a divider of 1, 2, 4, or 8. SMCLK can be used as the clock signal for Timer A and Timer B

5) **What are the functions of the Watchdog timer? What is its primary function, and what can it be configured to do?**

- The MSP430 Watchdog Timer's primary function is to perform a controlled-system restart after a software problem occurs.
- WDTCNT is a 32-bit up counter not accessible by software.
https://www.ti.com/lit/ug/slau399f/slau399f.pdf?ts=1603941410146&ref_url=https%253A%252F%252Fwww.google.com%252F

6) **What is the purpose of different registers and values like UCA0RXIFG, UCA0TXIFG, UCA0TXBUF, UCA0RXBUF, UCA0TXIE, UCA0RXIE in the code? What does it mean when they are set?**

- UCA0RXIFG – When a character is received in the UCA0RXBUF register, the UCA0RXIFG bit is set.
- UCA0TXIFG – interrupt flag is set by the transmitter when the UCA0TXBUF is ready to accept a new character.
- UCA0TXBUF – Register to transmit characters out to the command line.
- UCA0RXBUF – Register to hold received character
- UCA0TXIE – Transmit buffer interrupt enable bit
- UCA0RXIE – Receive buffer interrupt enable bit

7) **What is the size of a string or buffer created by sprintf? (Hint: Do not forget about the null character '\0')**

The size of the buffer should be large enough to contain the entire resulting string (see snprintf for a safer version).
Note the function's return type is `int` - it returns the length of the converted string.
http://cplusplus.com/reference/cstdio/sprintf/
https://www.educative.io/edpresso/how-to-use-the-sprintf-method-in-c

8) **Know the difference between Putty/MobaXterm and UAH Serial App.**

- Putty – Can only display ASCII characters and can only receive characters at 8-bit size.
- UAH Serial App – Allows us to view the different data types (int, float, etc.). This app translates serial packets that are sent to it, and it can graphically represent the data versus time. The UAH Serial Application expects a packet that has a 1-byte header followed by the data followed by an optional checksum.
- Moba X Term - remote computing.

9) Be able to describe the functionality of a given serial communication code snippet.

Pull up the Demo Code.

10) What is the difference between Synchronous and Asynchronous communication? What configuration needs to be shared between the nodes for asynchronous communication? How do we do that in MSP430 (Hint: divider and modulation)? What happens if the modulation value is wrong?

- Asynchronous – exchange of data between two or more parties without the requirement for all the recipients to respond immediately
- Since the two devices do not share a clock signal, there should be an agreement between the devices on the speed of the communication before the actual interface starts.
- Synchronous – two or more people exchange information in real-time and must share a common clock source.
- Modulation value wrong – the fraction part of the division of clock frequency by the baud rate will accumulate and eventually make the two devices unable to communicate.

```
P3SEL |= BIT3 + BIT4;    // Set USCI_A0 RXD/TXD to receive/transmit data
UCA0CTL1 |= UCSWRST;     // Set software reset during initialization
UCA0CTL0 = 0;            // USCI_A0 control register
UCA0CTL1 |= UCSSEL_2;    // Clock source SMCLK

UCA0BR0 = 0x09;          // 1048576 Hz  / 115200 lower byte
UCA0BR1 = 0x00;          // upper byte
UCA0MCTL |= UCBRS0;      // Modulation (UCBRS0=0x01, UCOS16=0)

UCA0CTL1 &= ~UCSWRST;    // Clear software reset to initialize USCI state mac
```

11.)

3.3 UART Baud Rate Generation

The USCI baud rate generator can produce standard baud rates from non-standard source frequencies. It provides two modes of operation: low-frequency mode (UCOS16 = 0) and over-sampling mode (UCOS16 = 1).

The low-frequency mode allows generation of baud rates from low frequency clock sources that reduce energy consumed by the communication interface. For example, we may have F_{BAUD}=9600 bps, and the source clock is BRCLK=ACLK= 32,768 Hz. By dividing 32,768 with 9,600 we get N=3.41. The challenge is that the baud rate divider cannot use fractions. Instead we initialize the baud rate registers UCBRx = INT(N) = 3, and the UCBRSx field UCBRSx = round((N – INT(N))*8)=3. THe UCBRSx 3-bit field controls the second modulation stage. The way this works is as follows: 5 bits (or 8 – UCBRSx bits) will have duration of 3 source clock periods (BRCLK) and 3 bits (UCBRSx bits) will have duration of 4 (N+1 in general) source clock periods, BRCLK, providing an average to be close to 3.41. Thus, some bits during transmission take 3 BRCLK periods and some take 4 BRCLK periods. The duration is modulated in such a way to minimize the error in communication

from the targeted bit rate for each bit period. Figure 6 shows common combinations of clock sources and baud rates and how to set the baud rate control registers.

(the clock source on hz) / (the buad rate you want) = x = UCA0BR0 lower and UCA0BR1 upper

1 048 576	57600	18
1 048 576	115200	9

```
UCA0BR0 = 0x09;              // 1048576 Hz  / 115200 lower byte
UCA0BR1 = 0x00;              // upper byte
```