1. Preparation task

f_{CPU} = 16 MHz

$$t_{ovf} = \frac{1}{f_{CPU}} \cdot 2^n \cdot N =$$

Module	Number of bits	1	8	32	64	128	256	1024
Timer/Counter0	8	16 u	128 u	--	1,024 m	--	4,096 m	16,38 m
Timer/Counter1	16	4,096 m	32,8 m	--	262,1 m	--	1049 m	4194 m
Timer/Counter2	8	16 u	128 u	512 u	1,024 m	2048 u	4,096 m	16,38 m

2. Timer library

Table with ATmega328P counters

Module	Operation	I/O register(s)	Bit(s)
Timer/Counter0	Prescaler 8-bit data value Overflow interrupt enable	TCCR0B TCNT0 TIMSK0	CS02, CS01, CS00 (000: stopped, 001: 0, 010: 8, 011: 64, 100: 256, 101: 1024) TCNT0[7:0] TOIE0 (1: enable, 0: disable)
Timer/Counter1	Prescaler 16-bit data value Overflow interrupt enable	TCCR1B TCNT1H, TCNT1L TIMSK1	CS12, CS11, CS10 (000: stopped, 001: 1, 010: 8, 011: 64, 100: 256, 101: 1024) TCNT1[15:0] TOIE1 (1: enable, 0: disable)
Timer/Counter2	Prescaler 8-bit data value Overflow interrupt enable	TCCR2B TCNT2 TIMSK2	CS22, CS21, CS20 (000: stopped, 001: 0, 010: 8, 011: 32, 100: 64, 101: 128, 110: 256, 111: 1024) TCNT2[7:0] TOIE2 (1: enable, 0: disable)

Table with ATmega328P selected interrupt sources

Program address	Source	Vector name	Description
0x0000	RESET	`--`	Reset of the system
0x0002	INT0	`INT0_vect`	External interrupt request number 0
0x0004	INT1	`INT1_vect`	External interrupt request number 1
0x0006	PCINT0	`PCINT0_vect`	Pin Change Interrupt Request 0
0x0008	PCINT1	`PCINT1_vect`	Pin Change Interrupt Request 1
0x000A	PCINT2	`PCINT2_vect`	Pin Change Interrupt Request 2
0x000C	WDT	`WDT_vect`	Watchdog Time-out Interrupt
0x0012	TIMER2_OVF	`TIM2_OVF_vect`	Overflow of Timer/Counter2 value
0x0018	TIMER1_COMPB	`TIMER1_COMPB_vect`	Compare match between Timer/Counter1 value and channel B compare value
0x001A	TIMER1_OVF	`TIMER1_OVF_vect`	Overflow of Timer/Counter1 value
0x0020	TIMER0_OVF	`TIM0_OVF_vect`	Overflow of Timer/Counter0 value
0x0024	USART_RX	`USART_RX_vect`	USART Rx Complete
0x002A	ADC	`ADC_vect`	ADC Conversion Complete
0x0030	TWI	`TWI_vect`	2-wire Serial Interface

timer.h:

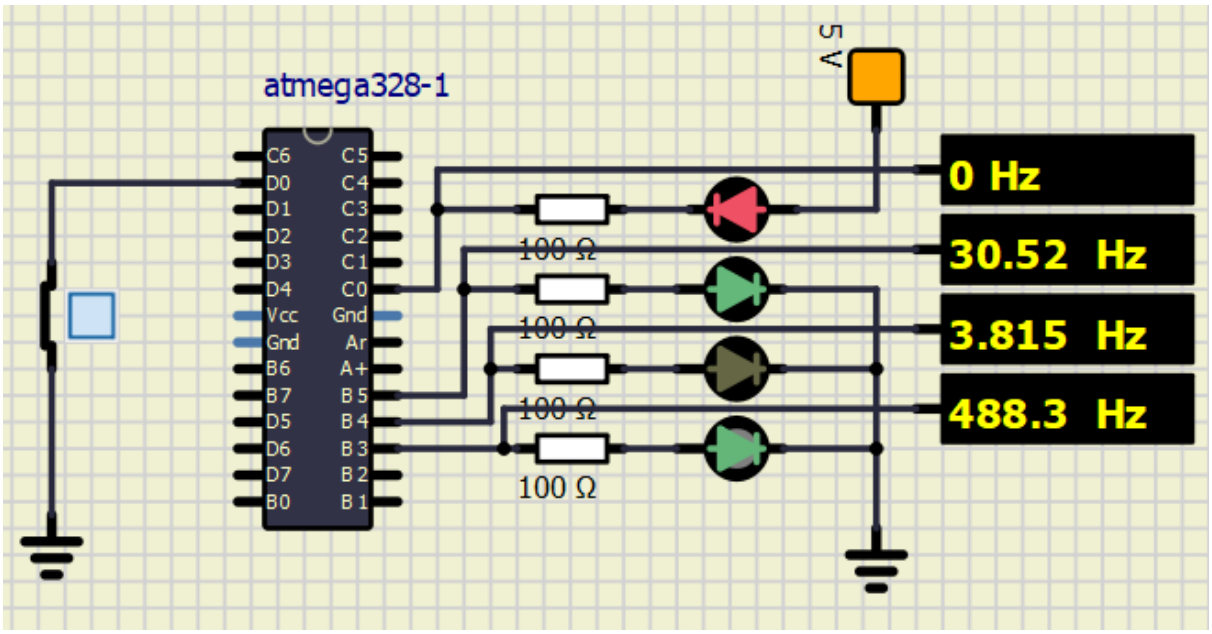
```
#ifndef TIMER_H
#define TIMER_H

/* Includes ------------------------------------------------------------*/
#include <avr/io.h>

/* Defines -------------------------------------------------------------*/
// @note  F_CPU = 16 MHz
// @brief Defines prescaler CPU frequency values for Timer/Counter0.
#define TIM0_stop()             TCCR0B &= ~((1<<CS02) | (1<<CS01) | (1<<CS00));
#define TIM0_overflow_16us()    TCCR0B &= ~((1<<CS02) | (1<<CS01)); TCCR0B |= (1<<CS00);
#define TIM0_overflow_128us()   TCCR0B &= ~((1<<CS02) | (1<<CS00)); TCCR0B |= (1<<CS01);
#define TIM0_overflow_1ms()       TCCR0B &= ~(1<<CS02); TCCR0B |= (1<<CS01) | (1<<CS00);
#define TIM0_overflow_4ms()     TCCR0B &= ~((1<<CS01) | (1<<CS00)); TCCR0B |= (1<<CS02);
#define TIM0_overflow_16ms()    TCCR0B &= ~(1<<CS01); TCCR0B |= (1<<CS02) | (1<<CS00);

// @brief Defines prescaler CPU frequency values for Timer/Counter1.
#define TIM1_stop()             TCCR1B &= ~((1<<CS12) | (1<<CS11) | (1<<CS10));
#define TIM1_overflow_4ms()     TCCR1B &= ~((1<<CS12) | (1<<CS11)); TCCR1B |= (1<<CS10);
#define TIM1_overflow_33ms()    TCCR1B &= ~((1<<CS12) | (1<<CS10)); TCCR1B |= (1<<CS11);
#define TIM1_overflow_262ms()   TCCR1B &= ~(1<<CS12); TCCR1B |= (1<<CS11) | (1<<CS10);
#define TIM1_overflow_1s()      TCCR1B &= ~((1<<CS11) | (1<<CS10)); TCCR1B |= (1<<CS12);
#define TIM1_overflow_4s()      TCCR1B &= ~(1<<CS11); TCCR1B |= (1<<CS12) | (1<<CS10);

// @brief Defines prescaler CPU frequency values for Timer/Counter2.
#define TIM2_stop()             TCCR2B &= ~((1<<CS22) | (1<<CS21) | (1<<CS20));             // 000
#define TIM2_overflow_16us()    TCCR2B &= ~((1<<CS22) | (1<<CS21)); TCCR2B |= (1<<CS20);    // 001
#define TIM2_overflow_128us()   TCCR2B &= ~((1<<CS22) | (1<<CS20)); TCCR2B |= (1<<CS21);    // 010
#define TIM2_overflow_512us()     TCCR2B &= ~(1<<CS22); TCCR2B |= (1<<CS21) | (1<<CS20);    // 011
#define TIM2_overflow_1ms()     TCCR2B &= ~((1<<CS21) | (1<<CS20)); TCCR2B |= (1<<CS22);    // 100
#define TIM2_overflow_2ms()       TCCR2B &= ~(1<<CS21); TCCR2B |= (1<<CS22) | (1<<CS20);    // 101
#define TIM2_overflow_4ms()       TCCR2B &= ~(1<<CS20); TCCR2B |= (1<<CS22) | (1<<CS21);    // 110
#define TIM2_overflow_16ms()    TCCR2B |= (1<<CS22) | (1<<CS21) | (1<<CS20);               // 111

/**
 * @brief Defines interrupt enable/disable modes for Timer/Counter0.
 */
#define TIM0_overflow_interrupt_enable()    TIMSK0 |= (1<<TOIE0);
#define TIM0_overflow_interrupt_disable()   TIMSK0 &= ~(1<<TOIE0);
/**
 * @brief Defines interrupt enable/disable modes for Timer/Counter1.
 */
#define TIM1_overflow_interrupt_enable()    TIMSK1 |= (1<<TOIE1);
#define TIM1_overflow_interrupt_disable()   TIMSK1 &= ~(1<<TOIE1);
/**
 * @brief Defines interrupt enable/disable modes for Timer/Counter2.
 */
#define TIM2_overflow_interrupt_enable()    TIMSK2 |= (1<<TOIE2);
#define TIM2_overflow_interrupt_disable()   TIMSK2 &= ~(1<<TOIE2);

#endif
```

main.c:

```
/*****************************************************************
 *
 * Control LEDs using functions from GPIO and Timer libraries. Do not
 * use delay library any more.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2018-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 *****************************************************************/

/* Defines -------------------------------------------------------*/
#define LED_D1  PB5
#define LED_D2  PB4
#define LED_D3  PB3

#define LED_RED             PC0
#define BTN                 PD0
#define BLINK_DELAY 500
#define F_CPU 16000000      // CPU frequency in Hz required for delay

/* Includes ------------------------------------------------------*/
#include <avr/io.h>         // AVR device-specific IO definitions
#include <util/delay.h>
#include <avr/interrupt.h>  // Interrupts standard C library for AVR-GCC
#include "gpio.h"           // GPIO library for AVR-GCC
#include "timer.h"          // Timer library for AVR-GCC

/* Function definitions ------------------------------------------*/
/**
 * Main function where the program execution begins. Toggle three LEDs
 * on Multi-function shield with internal 8- and 16-bit timer modules.
 */
int main(void)
{
    /* Configuration of three LEDs */
    GPIO_config_output(&DDRB, LED_D1);
    GPIO_write_low(&PORTB, LED_D1);

    GPIO_config_output(&DDRB, LED_D2);
    GPIO_write_low(&PORTB, LED_D2);

    GPIO_config_output(&DDRB, LED_D3);
    GPIO_write_low(&PORTB, LED_D3);

    /* RED LED */
    GPIO_config_output(&DDRC, LED_RED);
    GPIO_write_high(&PORTC, LED_RED);
       /* push button */
       GPIO_config_input_pullup(&DDRD, BTN);

    /* Configuration of 8-bit Timer/Counter0 */
    TIM0_overflow_16ms();
    TIM0_overflow_interrupt_enable();

    /* Configuration of 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */
    TIM1_overflow_262ms();
    TIM1_overflow_interrupt_enable();
```

```
    /* Configuration of 8-bit Timer/Counter2 */
    TIM2_overflow_4ms();
    TIM2_overflow_interrupt_enable();

    // Enables interrupts by setting the global interrupt mask
    sei();

    // Infinite loop
    while (1)
    {
        /* Empty loop. All subsequent operations are performed exclusively
         * inside interrupt service routines ISRs */
            // Pause several milliseconds
        _delay_ms(BLINK_DELAY);

        if(!GPIO_read(&PIND,BTN))
        {
               GPIO_toggle(&PORTC,LED_RED);
        }

       }

    // Will never reach this
    return 0;
}

/* Interrupt service routines ----------------------------------------*/
/**
 * ISR starts when Timer/Counter0 overflows. Toggle LED D1 on
 * Multi-function shield. */
ISR(TIMER0_OVF_vect)
{
    GPIO_toggle(&PORTB, LED_D1);
}
/**
 * ISR starts when Timer/Counter1 overflows. Toggle LED D2 on
 * Multi-function shield. */
ISR(TIMER1_OVF_vect)
{
    GPIO_toggle(&PORTB, LED_D2);
}
/**
 * ISR starts when Timer/Counter2 overflows. Toggle LED D3 on
 * Multi-function shield. */
ISR(TIMER2_OVF_vect)
{
    GPIO_toggle(&PORTB, LED_D3);
}
```

simulIDE circuit:



difference between a common C function and interrupt service routine:

Common C function has to check required condition in every cycle of its loop.

Iterrupt service routine works like this: periphery sends interrupt request to the main function. function stops, completes the task requested from periphery. Then continues, where it stopped.

3. PWN submit:

Table with PWM channels of ATmega328P:

Module	Description	MCU pin	Arduino pin
Timer/Counter0	OC0A	PD6	6
	OC0B	PD5	5
Timer/Counter1	OC1A	PB1	9
	OC1B	PB2	10
Timer/Counter2	OC2A	PB3	11
	OC2B	PD3	3

fast PWM mode:

Option of making pulse width modulated signal. A digital signal with changeable width of pulse (changeable duty). Its using moving with interrupt flags on TCNTn signal.

Usable for regulating light intensity of LEDs.