Session 4

-

Interrupts, timers

Author: Guillermo Cortés Orellana

Teacher: Tomáš Frýza

BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Lab assignment

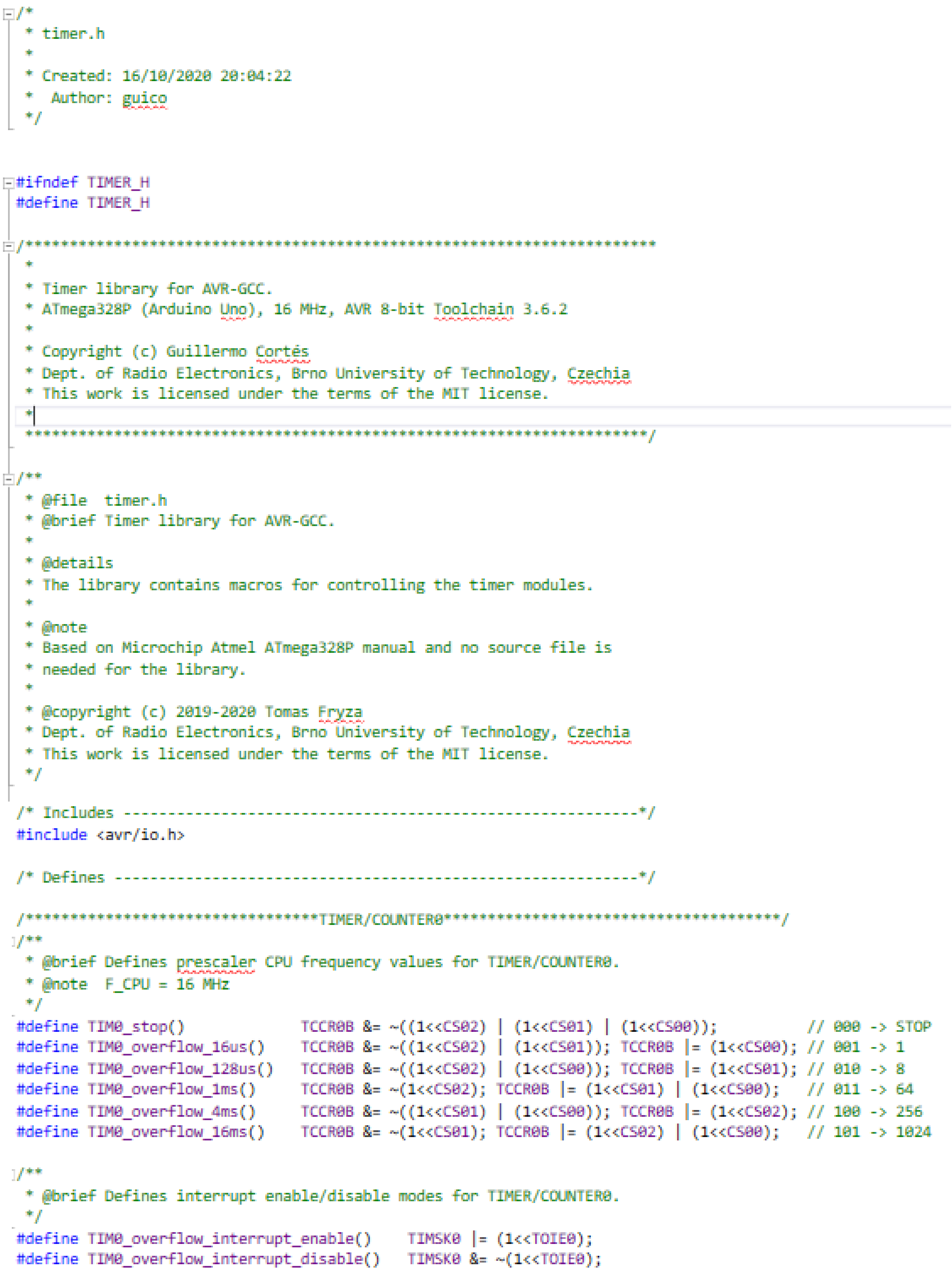
1. **Preparation tasks**

- Table with overflow times

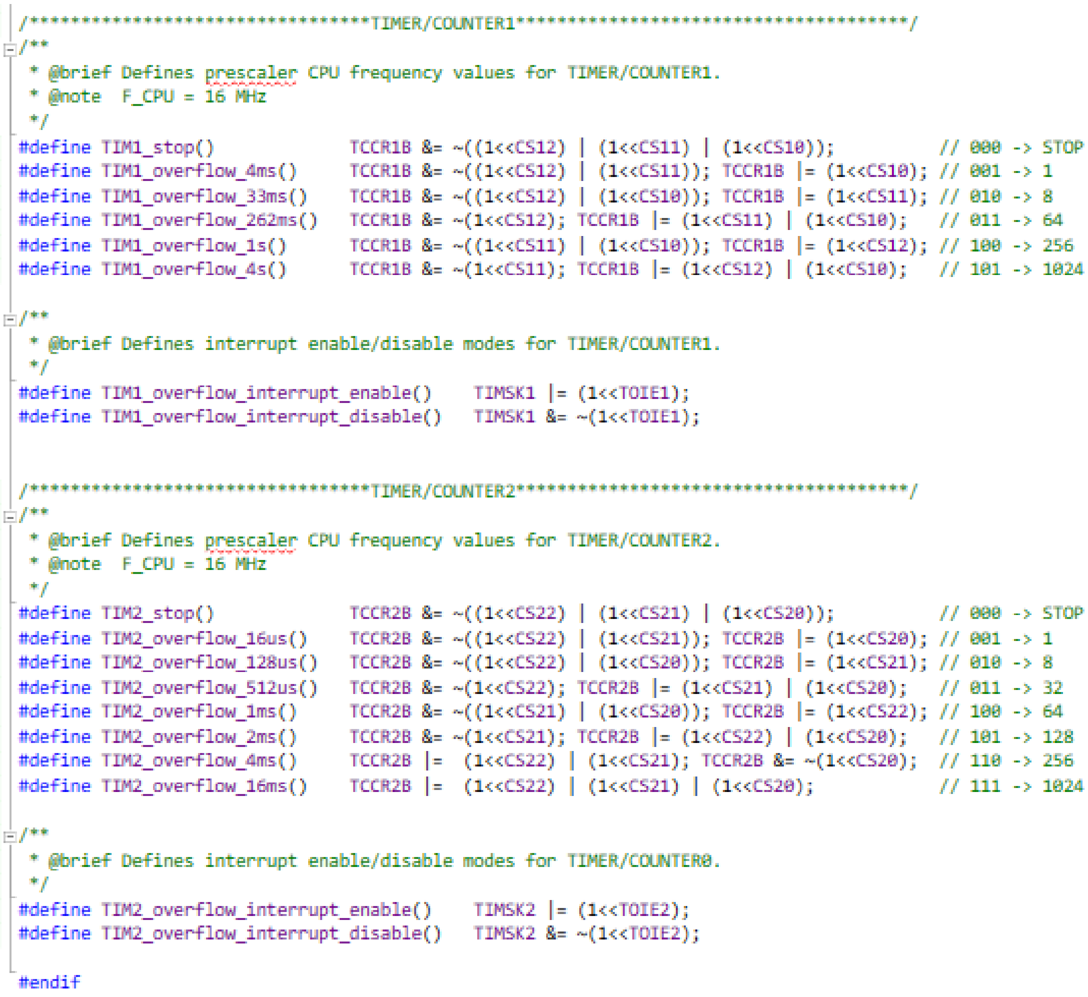
Module	**Number of bits**	**1**	**8**	**32**	**64**	**128**	**256**	**1024**
Timer/Counter0	8	16u	128u	-	1024u	-	4096u	16384u
Timer/Counter1	16	4096	32768u	-	262144u	-	1’04	4’19
Timer/Counter2	8	16u	128u	-	1024u	-	4096u	16384u

2. Timer library

- Listing of library header file ***timer.h***



```
/*
 * timer.h
 *
 * Created: 16/10/2020 20:04:22
 *  Author: guico
 */


#ifndef TIMER_H
#define TIMER_H

/***********************************************************************
 *
 * Timer library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) Guillermo Cortés
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 **********************************************************************/

/**
 * @file  timer.h
 * @brief Timer library for AVR-GCC.
 *
 * @details
 * The library contains macros for controlling the timer modules.
 *
 * @note
 * Based on Microchip Atmel ATmega328P manual and no source file is
 * needed for the library.
 *
 * @copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 */
/* Includes ----------------------------------------------------------*/
#include <avr/io.h>

/* Defines -----------------------------------------------------------*/

/*********************************TIMER/COUNTER0*************************************/
/**
 * @brief Defines prescaler CPU frequency values for TIMER/COUNTER0.
 * @note  F_CPU = 16 MHz
 */
#define TIM0_stop()           TCCR0B &= ~((1<<CS02) | (1<<CS01) | (1<<CS00));          // 000 -> STOP
#define TIM0_overflow_16us()   TCCR0B &= ~((1<<CS02) | (1<<CS01)); TCCR0B |= (1<<CS00); // 001 -> 1
#define TIM0_overflow_128us()  TCCR0B &= ~((1<<CS02) | (1<<CS00)); TCCR0B |= (1<<CS01); // 010 -> 8
#define TIM0_overflow_1ms()    TCCR0B &= ~(1<<CS02); TCCR0B |= (1<<CS01) | (1<<CS00);   // 011 -> 64
#define TIM0_overflow_4ms()    TCCR0B &= ~((1<<CS01) | (1<<CS00)); TCCR0B |= (1<<CS02); // 100 -> 256
#define TIM0_overflow_16ms()   TCCR0B &= ~(1<<CS01); TCCR0B |= (1<<CS02) | (1<<CS00);   // 101 -> 1024

/**
 * @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);
```



```c
/*********************************TIMER/COUNTER1**************************************/
/**
 * @brief Defines prescaler CPU frequency values for TIMER/COUNTER1.
 * @note  F_CPU = 16 MHz
 */
#define TIM1_stop()             TCCR1B &= ~((1<<CS12) | (1<<CS11) | (1<<CS10));          // 000 -> STOP
#define TIM1_overflow_4ms()     TCCR1B &= ~((1<<CS12) | (1<<CS11)); TCCR1B |= (1<<CS10); // 001 -> 1
#define TIM1_overflow_33ms()    TCCR1B &= ~((1<<CS12) | (1<<CS10)); TCCR1B |= (1<<CS11); // 010 -> 8
#define TIM1_overflow_262ms()   TCCR1B &= ~(1<<CS12); TCCR1B |= (1<<CS11) | (1<<CS10);   // 011 -> 64
#define TIM1_overflow_1s()      TCCR1B &= ~((1<<CS11) | (1<<CS10)); TCCR1B |= (1<<CS12); // 100 -> 256
#define TIM1_overflow_4s()      TCCR1B &= ~(1<<CS11); TCCR1B |= (1<<CS12) | (1<<CS10);   // 101 -> 1024

/**
 * @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);


/*********************************TIMER/COUNTER2**************************************/
/**
 * @brief Defines prescaler CPU frequency values for TIMER/COUNTER2.
 * @note  F_CPU = 16 MHz
 */
#define TIM2_stop()             TCCR2B &= ~((1<<CS22) | (1<<CS21) | (1<<CS20));          // 000 -> STOP
#define TIM2_overflow_16us()    TCCR2B &= ~((1<<CS22) | (1<<CS21)); TCCR2B |= (1<<CS20); // 001 -> 1
#define TIM2_overflow_128us()   TCCR2B &= ~((1<<CS22) | (1<<CS20)); TCCR2B |= (1<<CS21); // 010 -> 8
#define TIM2_overflow_512us()   TCCR2B &= ~(1<<CS22); TCCR2B |= (1<<CS21) | (1<<CS20);   // 011 -> 32
#define TIM2_overflow_1ms()     TCCR2B &= ~((1<<CS21) | (1<<CS20)); TCCR2B |= (1<<CS22); // 100 -> 64
#define TIM2_overflow_2ms()     TCCR2B &= ~(1<<CS21); TCCR2B |= (1<<CS22) | (1<<CS20);   // 101 -> 128
#define TIM2_overflow_4ms()     TCCR2B |=  (1<<CS22) | (1<<CS21); TCCR2B &= ~(1<<CS20);  // 110 -> 256
#define TIM2_overflow_16ms()    TCCR2B |=  (1<<CS22) | (1<<CS21) | (1<<CS20);            // 111 -> 1024

/**
 * @brief Defines interrupt enable/disable modes for TIMER/COUNTER0.
 */
#define TIM2_overflow_interrupt_enable()    TIMSK2 |= (1<<TOIE2);
#define TIM2_overflow_interrupt_disable()   TIMSK2 &= ~(1<<TOIE2);

#endif
```

You can find the code on my GitHub:

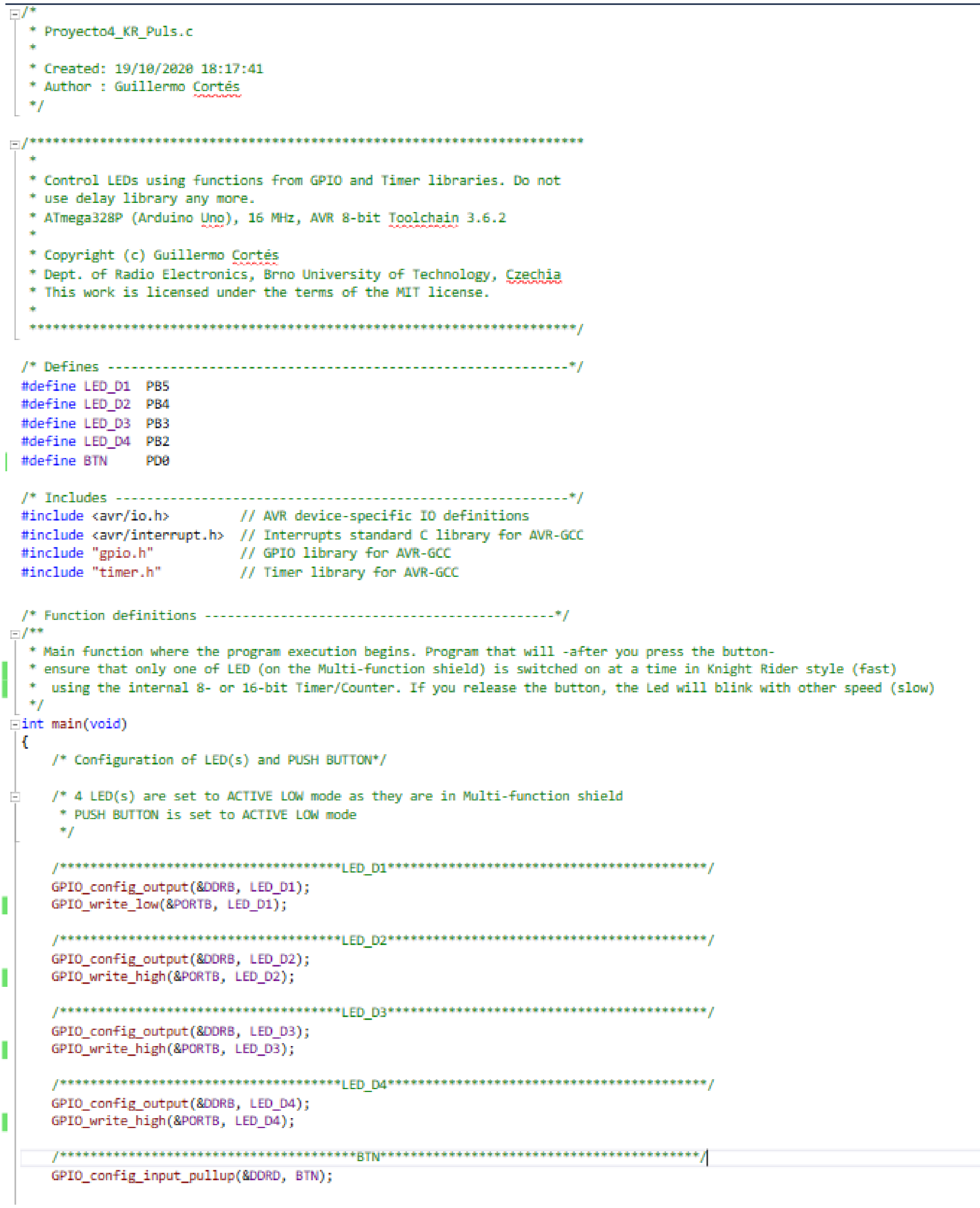
https://github.com/GuicoRM/Digital-Electronics-2

- Table with ATmega328P selected interrupts sources

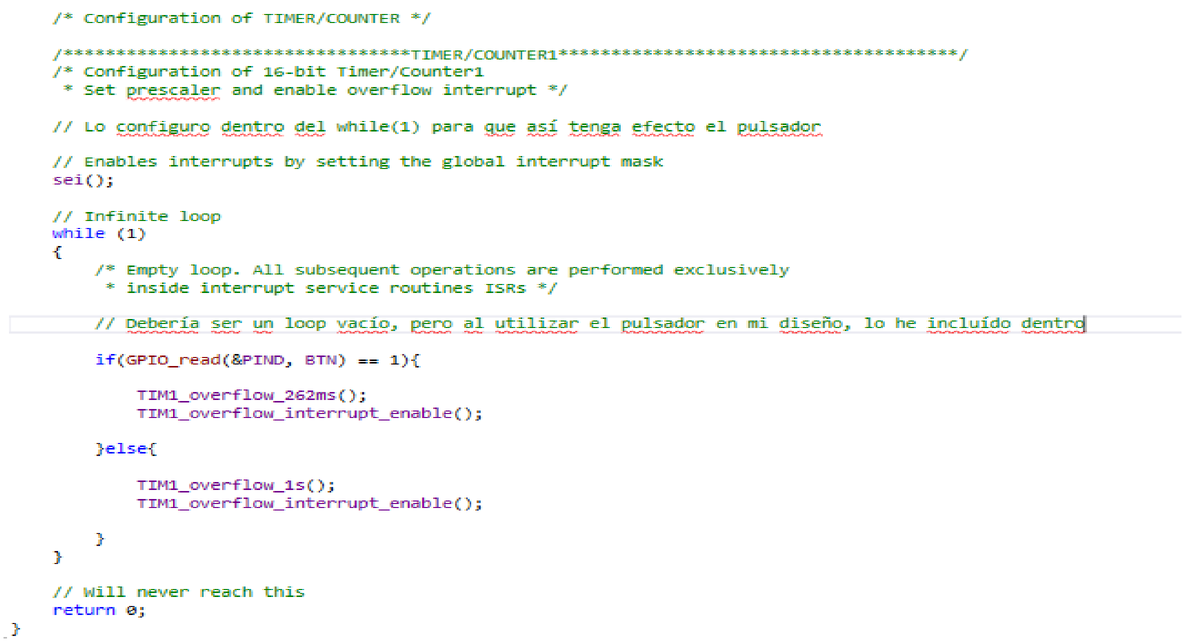
Program address	Source	Vector name	Description
0x0000	RESET	-	Reset of the system
0x0002	INT0	INT0_vect	External interrupt request number 0
0x0004	INT1	INT1_vect	External interrupt request number 1
0x0006	PCINT0	PCINT0_vect	Pin change interrupt request 0
0x0008	PCIN1	PCINT1_vect	Pin change interrupt request 1
0x000A	PCINT2	PCINT2_vect	Pin change interrupt request 2
0x000c	WDT	WDT_vect	Watchdog time-out interrupt
0x0012	TIMER2_OVF	TIMER2_OVF_vect	Overflow of Timer/Counter2 value
0x0018	TIMER1_COMPB	TIMER1_COMPB_vect	Compare match between Timer/Counter1 value and cannel B compare value
0x001A	TIMER1_OVF	TIMER1_OVF_vect	Overflow of Timer/Counter1 value
0x0020	TIMER0_OVF	TIMER0_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

- Listing of the final application ***main.c***

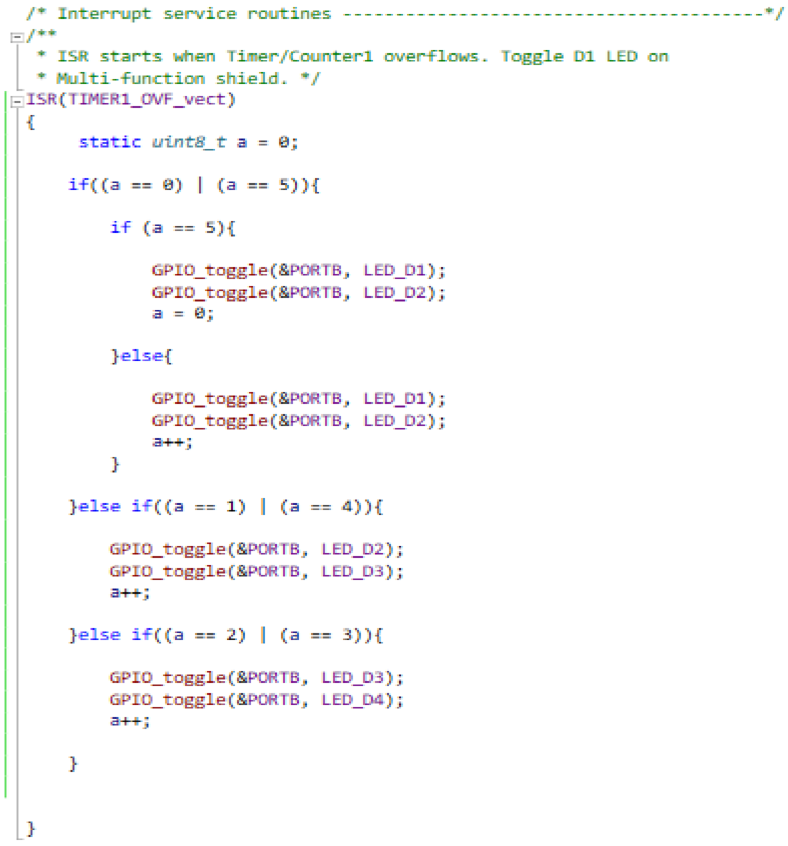
I got LEDS blink in Knight-Rider style using ISR and a push Button but with the configuration IN THE LOOP (which should be empty).



```
/*
 * Proyecto4_KR_Puls.c
 *
 * Created: 19/10/2020 18:17:41
 * Author : Guillermo Cortés
 */

/***********************************************************************
 *
 * 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) Guillermo Cortés
 * 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_D4  PB2
#define BTN     PD0

/* Includes ----------------------------------------------------------*/
#include <avr/io.h>         // AVR device-specific IO definitions
#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. Program that will -after you press the button-
 * ensure that only one of LED (on the Multi-function shield) is switched on at a time in Knight Rider style (fast)
 *  using the internal 8- or 16-bit Timer/Counter. If you release the button, the Led will blink with other speed (slow)
 */
int main(void)
{
    /* Configuration of LED(s) and PUSH BUTTON*/

    /* 4 LED(s) are set to ACTIVE LOW mode as they are in Multi-function shield
     * PUSH BUTTON is set to ACTIVE LOW mode
     */

    /*************************************LED_D1*******************************************/
    GPIO_config_output(&DDRB, LED_D1);
    GPIO_write_low(&PORTB, LED_D1);

    /*************************************LED_D2*******************************************/
    GPIO_config_output(&DDRB, LED_D2);
    GPIO_write_high(&PORTB, LED_D2);

    /*************************************LED_D3*******************************************/
    GPIO_config_output(&DDRB, LED_D3);
    GPIO_write_high(&PORTB, LED_D3);

    /*************************************LED_D4*******************************************/
    GPIO_config_output(&DDRB, LED_D4);
    GPIO_write_high(&PORTB, LED_D4);

    /***************************************BTN******************************************/
    GPIO_config_input_pullup(&DDRD, BTN);
```



```
    /* Configuration of TIMER/COUNTER */

    /**********************************TIMER/COUNTER1***************************************/
    /* Configuration of 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */

    // Lo configuro dentro del while(1) para que así tenga efecto el pulsador

    // 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 */

        // Debería ser un loop vacío, pero al utilizar el pulsador en mi diseño, lo he incluído dentro

        if(GPIO_read(&PIND, BTN) == 1){

            TIM1_overflow_262ms();
            TIM1_overflow_interrupt_enable();

        }else{

            TIM1_overflow_1s();
            TIM1_overflow_interrupt_enable();

        }
    }

    // Will never reach this
    return 0;
}
```



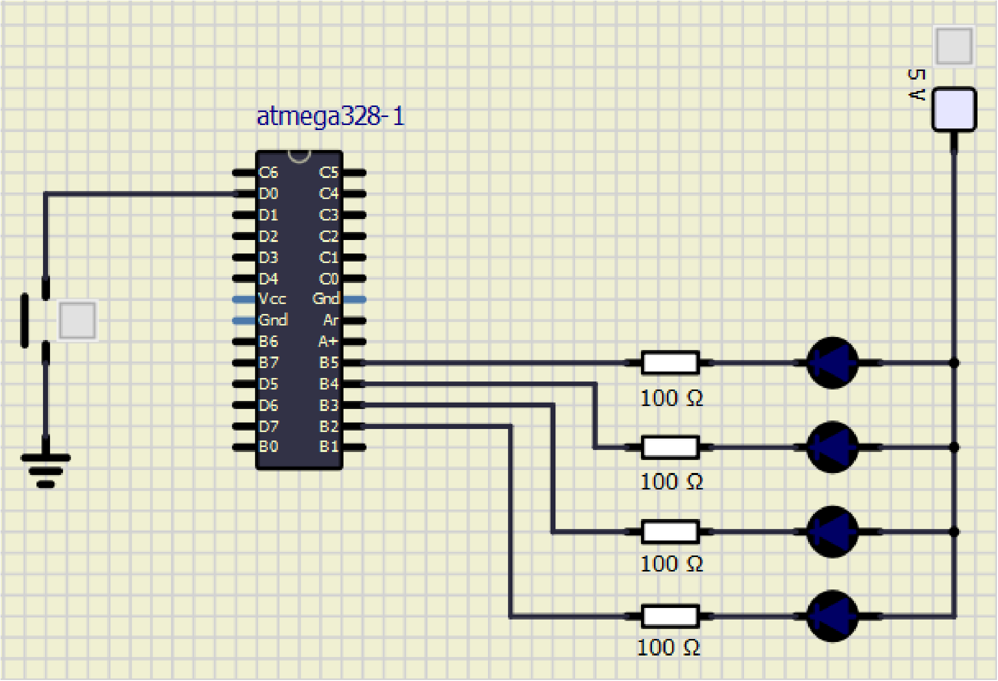
```
/* Interrupt service routines ----------------------------------------*/
/**
 * ISR starts when Timer/Counter1 overflows. Toggle D1 LED on
 * Multi-function shield. */
ISR(TIMER1_OVF_vect)
{
     static uint8_t a = 0;

    if((a == 0) | (a == 5)){

        if (a == 5){

            GPIO_toggle(&PORTB, LED_D1);
            GPIO_toggle(&PORTB, LED_D2);
            a = 0;

        }else{

            GPIO_toggle(&PORTB, LED_D1);
            GPIO_toggle(&PORTB, LED_D2);
            a++;
        }

    }else if((a == 1) | (a == 4)){

        GPIO_toggle(&PORTB, LED_D2);
        GPIO_toggle(&PORTB, LED_D3);
        a++;

    }else if((a == 2) | (a == 3)){

        GPIO_toggle(&PORTB, LED_D3);
        GPIO_toggle(&PORTB, LED_D4);
        a++;

    }


}
```

You can find the code on my GitHub:

https://github.com/GuicoRM/Digital-Electronics-2

- Screenshot of SimulIDE circuit

  <u>Note</u>: LEDS are ACTIVE LOW as ‘Multi-function shield’



- In your Word, describe the difference between a common C function and interrupt service routine

  - **C functions** are blocks of code (tools) designed by the programmer for a specific purpose. They are usually used in different parts of the code at a certain point in time. Meanwhile, **ISR (I**nterrupt **S**ervice **R**outine**)** are blocks of code that run asynchronously and therefore somehow interrupt the work of the preocessor.

3. **PWM**

- Table with PWM channels of ATmega328P

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

- Describe the behavior of Clear Timer on Compare and fast PWM modes

In Clear **Timer on Compare** or CTC mode (WGM02:0 = 2), the OCR0A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

On the other hand, **Fast Pulse Width Modulation or fast PWM** mode (WGM02:0 = 3 or 7) provides a high frequency PWM waveform generation option.