Bern University
of Applied Sciences

# Exercise - 07

## General-purpose Timers and Soft PWM

Andreas Habegger | Adrian Steiner
BTS3230 | Version 1.0.0 of 17.03.2025

Please be aware that the content is subject to change at any time. For the latest version, please check the website.

In this exercise you will learn the concepts of a **General Purpose Timer**. By combining the general-purpose timer with interrupts and events, you can implement a software based pulse-width modulation (PWM), for example.

This exercise is about a simple brightness controller to implement a headlight dimmer application. It is a continuation exercise, reusing some of the previous exercises.

### Objectives

- General Purpose Timers - Output Comparison
- Nested Vectored Interrupt Controller “NVIC
- EXTI” External Interrupt/Event Controller

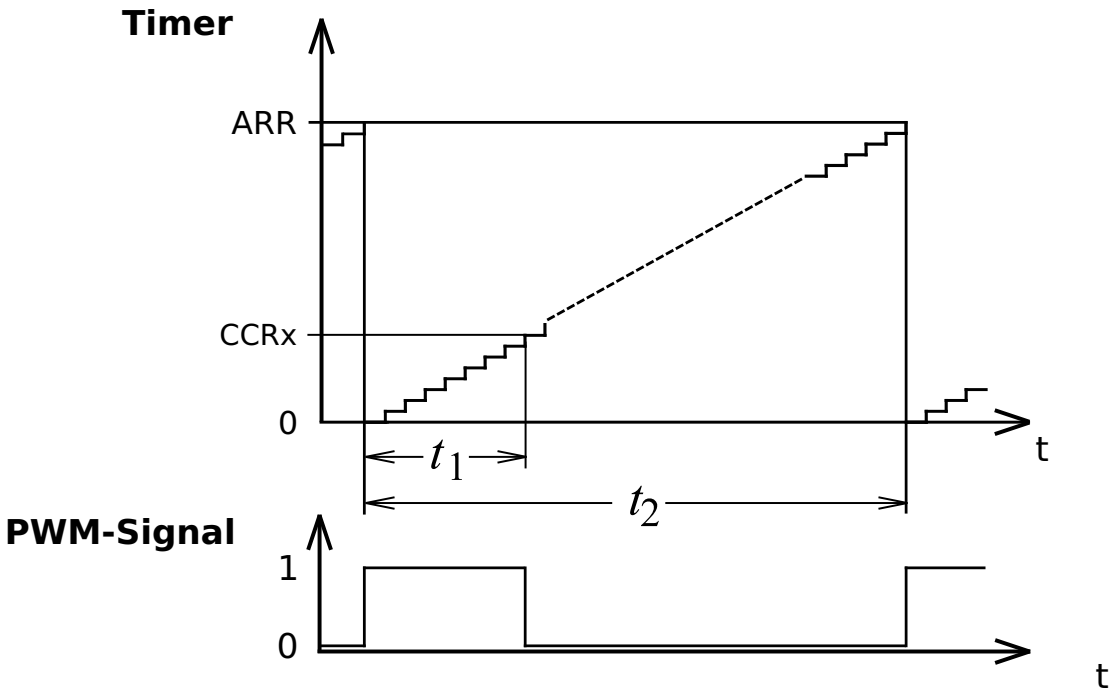
### Outcomes

- Understanding PWM signals with duty cycle
- Advantage of general purpose timers over simple timers
- Understanding the behaviour of the CCR for edge values

# Description

In the last exercise you learned how to use simple timers and their interrupts. Your blinking LED application used a basic timer as a time base. When the timer overflowed, the LED was toggled. So the LED was enabled and disabled for the duration of one cycle of the timer. A smaller value for the auto-reload register “ARR” would result in a higher switching frequency. However, the brightness would remain the same due to the constant on/off ratio. In this exercise you will learn how to implement a soft PWM with a variable on/off ratio. This ratio is called the duty cycle or duty factor.

**General-Purpose Timer with PWM**




PWM signal in relation to the timer registers

Although it would be possible to implement a soft PWM using a simple timer, we will look for a simpler solution. We will use a general purpose timer which introduces a new feature, the capture/compare register “CCR”. This register allows an event/interrupt to be generated when the counter register “CNT” matches the value in the capture/compare register “CCR”.

# Tasks

This exercise consists of two main tasks. First, the soft PWM, which uses a timer as a time base together with hardware interrupts. Second, the application code, which provides multiple dimming levels and a function to store dimming ratios.

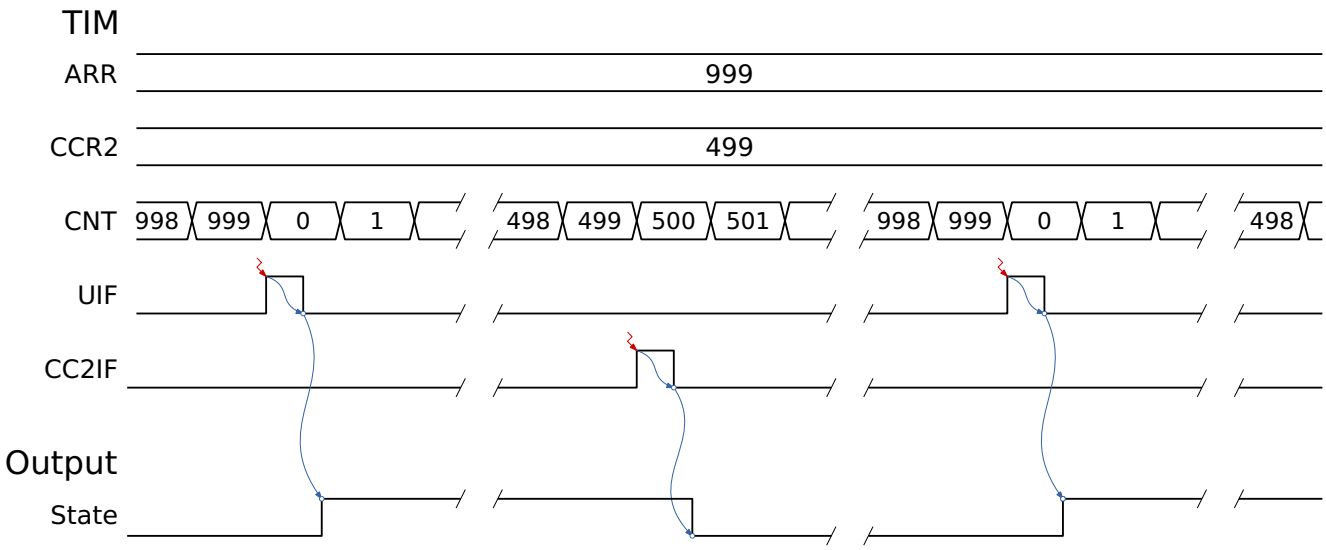
## Soft PWM – General Purpose Timer

To implement a soft PWM with a variable “duty”, use two different interrupts generated by a general purpose timer:

- Update: Counter overflow/underflow
- Output Compare

The output pulls high on an update interrupt and low on an output compare interrupt.

**Timing Diagram**




General Purpose Timer – Timing Diagram

**Configuration**

The initialisation of a General Purpose Timer includes the enabling of the interrupts “UIE” and “CC2IE” and the reload behaviour. The initial values for auto-reload “ARR” and capture/compare “CCR”, as well as the prescaler for scaling the timer’s internal clock, must be set to the custom values. The resulting frequency is calculated in the same way as for the base timers:

$$f_{\text{TIM}} = \frac{f_{APBx}}{(PSC + 1) * (ARR + 1)}$$

In general, the duty cycle is given as a ratio between the following values

$$Duty = \frac{CCR}{ARR}$$

**ISR of TIM3**

You need to distinguish between the two sources because the update and compare interrupts share the same IRQ.

### Example basic interrupt handler

```
void TIM3_IRQHandler(void) {
  if (READ_BIT(TIM3->SR, TIM_SR_UIF)) {
    CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
    /* CUSTOM CODE AT UPDATE EVENT*/
  }

  if (READ_BIT(TIM3->SR, TIM_SR_CC2IF)) {
    CLEAR_BIT(TIM3->SR, TIM_SR_CC2IF);
    /* CUSTOM CODE AT COMPARE EVENT*/
  }
}
```

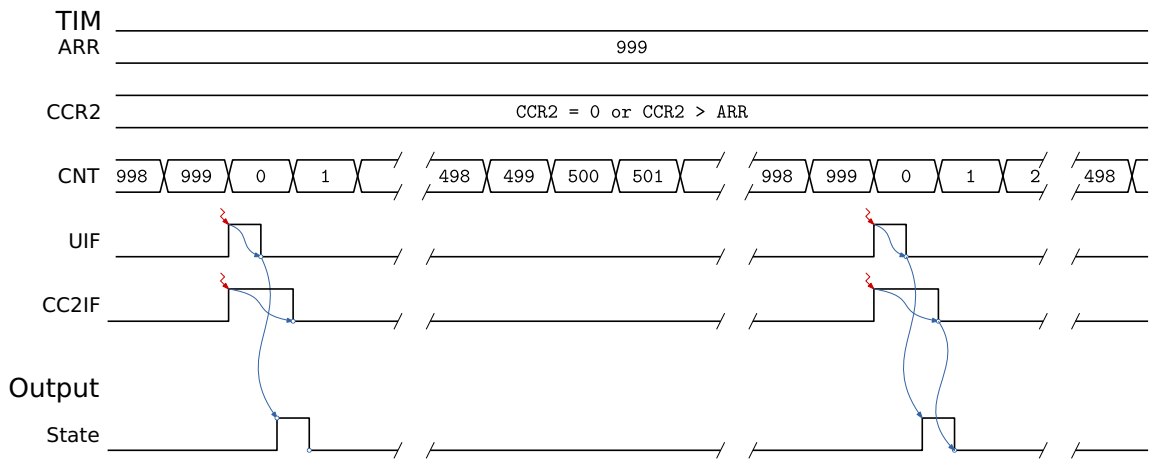
In both cases the flag must be cleared first. Depending on the interrupt source, the output is either high or low.

**Additional information**

General purpose timers offer much more functionality than simple timers, such as the Capture Compare Register (CCR). An event can be triggered when the CCR or ARR values match. Let's take a closer look at the constraints.

▸ **Case I - CCR = 0**

### CCR2 Timing Diagram CCR = 0 | CCR > ARR



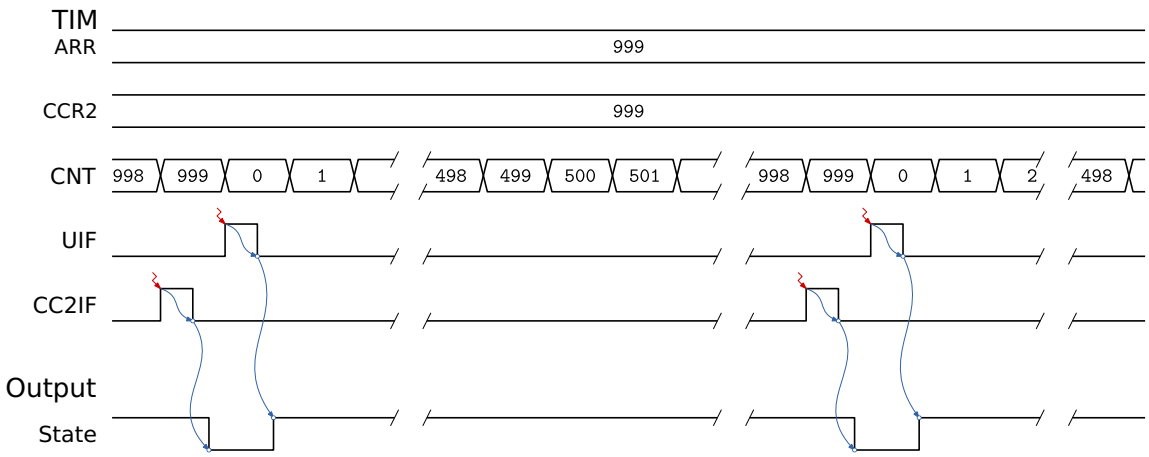

General Purpose Timer - Boundary CCR = 0 or CCR > ARR

The figure shows the case where the Capture/Compare Register (CCR) is zero. When the CCR value is zero, the CCR and the UE interrupt are fired at the same time. So the behaviour depends on your implementation. Since the same IRQ is sent by the timer for both interrupts, you must decide which flag to handle first. But even if you handle both interrupts correctly, the LED (of simulated headlight application) may glow because it has switched temporarily due to the double execution of the ISR.

This will show up as a short spike in the output. An additional condition can help to solve this problem.

- **Case II – CCR > ARR** If the “CCR” value is greater than the “ARR” value, the capture/compare interrupt will be triggered during the update event. You will therefore end up with the same problem as in the previous case (Case I).
- **Case III – CCR = ARR**

**CCR2 Timing Diagram CCR = ARR**




General Purpose Timer – Boundary CCR = ARR

If the “CCR” value matches the “ARR” value, the compare interrupts will fire one timer clock cycle earlier than the update event interrupt. This is slightly the opposite of the LED glowing, as it is not at its maximum brightness.

## Application

The application provides the functionality of a torch. A short press and release of the centre button turns the light (LED) on or off. The up and down buttons are used to increase or decrease the current brightness level. When the torch is activated, it always starts with the default brightness level, $100\,\%$. However, the user can override the current default brightness level by pressing and holding the centre switch for longer than $1\,\mathrm{s}$. This will save the current brightness level as the new default. The list below gives an overview of the required functionality:

- **On/Off switching**

  A short **press and release** action on the **center switch** allows the LED to be switched on and off. By default, the brightness is set to maximum.

- **Overwrite default**

  Store a new default brightness value. **Pressing and holding** the **center switch** for longer than $1\,\mathrm{s}$ will overwrite the default brightness when the LED is switched on. The new brightness level will be used as the new default when the LED is switched on.

- **Dimming**

A rising edge of the **up** or **down switch** will increase or decrease the brightness of the LED by $10\,\%$.

For the centre button you can use the same code as in the last exercise. This will give you the logic for a short press and release or hold event. The up and down switches can just be handled by a rising edge interrupt. There is no need to debounce these two signals. By reusing the code from the previous exercise, you already have the event handling structure.

## Implementation

The first step is to initialise the LED as you have done many times before. No special configuration is required for this exercise.

**Initialisation LED**

1. Initialise the RGB LEDs as general purpose outputs.
2. Write a function to set and clear the RGB LED (one channel would be required).

A general purpose timer is used to control the LED. For this exercise it does not matter which general purpose timer (TIM2-TIM5, TIM9-TIM14) is used.

**Configuration Timer**

3. Initialise a general purpose timer. Use a timer frequency around $1\,\mathrm{kHz}$ with a duty cycle of $100\,\%$.

**TIM3 Initialization**

```
void custom_TIM3_init(void) {
  /* Control Register 1 - Reset Value 0x0000 */
  SET_BIT(TIM3->CR1, TIM_CR1_URS);  // Only counter over-/underflow interrupt
  SET_BIT(TIM3->CR1, TIM_CR1_ARPE); // ARR is buffered

  /* DMA/Interrupt Enable Register - Reset Value 0x0000 */
  SET_BIT(TIM3->DIER, TIM_DIER_UIE);   // Update interrupt enable
  SET_BIT(TIM3->DIER, TIM_DIER_CC2IE); // Capture/Compare 2 interrupt enable

  /* Event Generation Register - Reset Value 0x0000 */
  SET_BIT(TIM3->EGR, TIM_EGR_CC2G); // Channel 2 enable interrupt request

  /* Capture/Compare Mode Register 1 - Reset Value 0x0000 */
  SET_BIT(TIM3->CCMR1, TIM_CCMR1_OC2PE); // Output compare 2 preload enable

  /* Prescaler - Reset Value 0x000 */
  WRITE_REG(TIM3->PSC, custom_TIM3_PRESCALER_VALUE); // Set prescaler value

  /* Auto-Reload Register - Reset Value 0x0000 */
  WRITE_REG(TIM3->ARR, custom_TIM3_ARR_VALUE); // Set ARR register

  /* Capture/Compare Register 1 - Reset Value 0x0000 */
  WRITE_REG(TIM3->CCR2, custom_TIM3_CCR2_VALUE); //  Set CCR2 register
}
```

4. Enable the interrupt on the NVIC for the corresponding general purpose timer.
5. Implement the interrupt handler of the used timer and toggle the LED on the corresponding event.

**Warning**

Do not forget the additional information about the boundary events of CCR and ARR values.

6. Test your implementation with several duty cycle values, including edge values. Make the correct operation visible with the sales.

The UP switch increases the duty cycle with a step size of $10\,\%$. The DOWN switch decreases the duty cycle with the same step size. A duty cycle of $0\,\%$ should not be possible.

**Dimmer**

7. Configure the EXTI controller to add an interrupt to the UP and DOWN switches.
8. Enable the interrupts used in the NVIC controller.

9. Implement the interrupt handler to increase and decrease the current duty cycle.

Enable and disable the timer to turn off the headlight instead of setting the PWM to $0\,\%$. When switched on, the fixture will illuminate at the default brightness. At first power up this value is $100\,\%$.

**Turn headlight on/off**

10. Configure the EXTI controller for the centre switch pin.
11. Enable the new interrupt in the NVIC.
12. Switch the headlights on or off on a “FALLING EDGE”.

Finally, the last desired functionality is implemented. The goal is to detect the duration of the centre switch button and after $1\,\mathrm{s}$ set the current brightness as the new default value. to set the current brightness as the new default value when the headlight is switched on.

**Default Brightness**

13. Add a timer that detects the duration of the centre switch pin.
14. Implement the following functionality on the centre switch pin:
    - State: Headlight OFF:

      Switch on the headlight on a “FALLING EDGE”
    - State: Headlight ON:
      - If: time pressed $>= 1\,\mathrm{s}$:

        Modify brightness (duty cycle) and save as new default value for off/on switching
      - Otherwise:

        Switch off the headlight

### Question

a. What effect does the timer frequency have? Try running the timer in a range from $10\,\mathrm{Hz}$ to $10\,\mathrm{kHz}$ and note the effect on the RGB LED at different duty cycles.

b. At which frequency do you no longer see flashing at a duty cycle of $10\,\%$?

### Solution

a. The effect of the frequency has a visible consequence. If the frequency is high enough, the dimming effect of the LED will be visible and not the actual flashing.

b. With a duty cycle of $10\,\%$ and a frequency of about $50\,\mathrm{Hz}$, no small flashing is visible.