### 2.4.2 Determining the PSC-value and ARR-value for dimmable LEDs

For dimmable LEDs, it appears that a PWM frequency ($F_{PWM}$) of 1,000Hz works well. One way to implement that PWM frequency is to use the largest ARR-value. Two techniques for finding that value will be described.

The first technique uses the maximum ARR-value (65,536). The PSC-value is varied, and FPWM is calculated. The results are shown below. A PSC-value of 3 results in an FPWM of 1098.6Hz, which is close enough to the objective of 1,000Hz.

| | | | (calculated) |
|---|---|---|---|
| **$F_{TIM}$** | **PSC_value** | **ARR_value** | **$F_{PWM}$** |
| 216000000 | 1 | 65536 | **3295.9** |
| 216000000 | 2 | 65536 | **1647.9** |
| 216000000 | 3 | 65536 | **1098.6** |
| 216000000 | 4 | 65536 | **824.0** |
| 216000000 | 5 | 65536 | **659.2** |

In the second technique, the PSC-value is varied, and the ARR-value is calculated [STMa, page 31]. The equation for the ARR-value is derived from the equations for $F_{PWM}$ and $F_{SCALED}$:

```
ARR = ( FTIM / (FPWM * (PSC + 1) ) - 1
```

A PSC-value of 3 cannot be used as it requires an ARR-value of 71,999, and the maximum ARR-value is 65,536. A PSC-value of 4 can be used, and it requires an ARR-value of 53,999. However, the earlier solution is preferable, with a PSC-value of 3 and an ARR-value of 65,536. By using a lower PSC value and higher ARR value, the duty-cycle calculation is more precise.

| | | | (calculated) |
|---|---|---|---|
| **$F_{TIM}$** | **PSC_value** | **$F_{PWM}$** | **ARR_value** |
| 216000000 | 1 | 1000 | 215999 |
| 216000000 | 2 | 1000 | 107999 |
| 216000000 | 3 | 1000 | 71999 |
| 216000000 | 4 | 1000 | 53999 |
| 216000000 | 5 | 1000 | 43199 |

## 3 How the dimmable LEDs are implemented

The book has an implementation of dimmable LEDs, in the Chapter 13 example programs. This section describes how the PWM frequency and the duty-cycle are configured. Speculative explanations are stated using terms such as, "appears to be...", or "apparently...".

The relevant code is in `pwmImplementation.c`. There, the function `PWMInit()` configures the needed hardware: clocks, timers, GPIOs, and LEDs. The brightness of the board's LEDs is set by the functions `SetBlueDuty()`, `SetRedDuty()`, and `SetGreenDuty()`.

The book's example programs are on GitHub. `pwmImplementation.c` is here:
https://github.com/PacktPublishing/Hands-On-RTOS-with-Microcontrollers/blob/master/Chapter_13/Src/pwmImplementation.c

The code for `PWMInit()` is probably generated by STM32CubeIDE. STM has a tutorial on how to use that IDE to generate code for a dimmable LED [STM20].

STM32 supports many different types of PWM output. The PWM type implemented in `pwmImplementation.c` appears to be: PWM mode 1, in edge-aligned mode, with up-counting. An STM tutorial lists the specific settings needed for that PWM mode [STM21b, page 16].

This line specifies PWM mode 1:
```
sConfig.OCMode = TIM_OCMODE_PWM1;
```
This line specifies up-counting:
```
TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
```
I couldn't find a way to select edge-aligned mode directly. It appears to get selected by specifying up-counting [STM21b, page 16].

These lines set the PSC and ARR registers:
```
uint32_t uhPrescalerValue = (uint32_t)((SystemCoreClock/2) / 21600000) - 1;
TimHandle.Init.Prescaler = uhPrescalerValue;
TimHandle.Init.Period = 65535;
```

How those lines set the PSC and ARR registers:
- `SystemCoreClock` is a global system variable. When this code runs it will have the system clock