**Networked Embedded Systems**

**Practicum 2: Timers**

**Group number: 8**

|  |  |  |
| --- | --- | --- |
| **Name** | **Student ID** | **Email** |
| Bernberger Sarah | k12112018 | sarahbernberger.sb@gmail.com |
| Grundner Simon | k12136610 | simon.grundner@gmail.com |

**09.05.2024**

Map of Content

[Theory Questions 3](#_Toc166212403)

[Task A: Blink an LED with Timer Module 7](#_Toc166212404)

[A.1. Calculations 7](#_Toc166212405)

[A.2. Implementation 8](#_Toc166212406)

[A.3. Discussion 9](#_Toc166212407)

[Task B: Vary the Brightness of an LED 10](#_Toc166212408)

[B.1. Calculations 11](#_Toc166212409)

[B.2. Implementation 12](#_Toc166212410)

[B.3. Results 14](#_Toc166212411)

[B.4. Discussion 14](#_Toc166212412)

[Task C: Count Pulses of a Blinking LED 15](#_Toc166212413)

[C.1. Implementation 16](#_Toc166212414)

[C.2. Discussion 18](#_Toc166212415)

[Index 19](#_Toc166212416)

[Figures 19](#_Toc166212417)

[Code Segments 20](#_Toc166212418)

[Tables 20](#_Toc166212419)

# Theory Questions

1. **Describe in your own words Timer Mode, PWM Mode, and Counter Mode. Which own use cases do you come up with?**

**Timer Mode**

A timer can be triggered from various clock sources, such as the Internal Clock (CK\_INT), an external input pin (TIx), an external trigger input (ETR) and from internal trigger inputs (ITRx).

Internal trigger inputs can be programmed as a source for a timer and allow the user to use one timer as a prescaler for another timer. For example, timer 2 is configured as a slave for Timer 3, which means that every time the Timer 3 triggers an update event (UEV) Timer 2 receives a pulse.

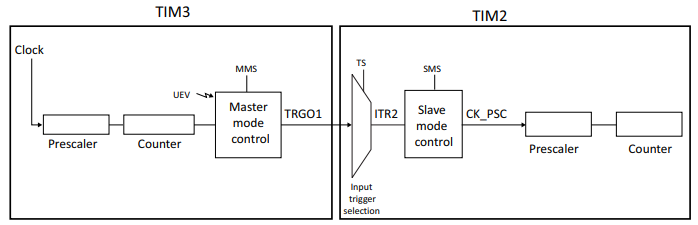
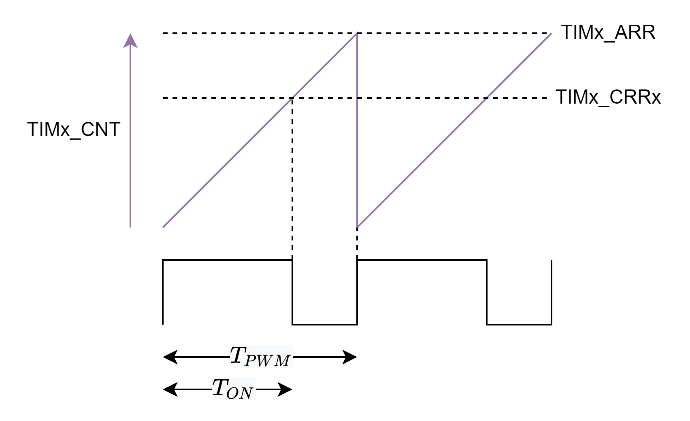


Figure 1: Master/Slave Timer Example

As mentioned, a use case for this operation would be to prescale another timer. This is needed when the user wants to implement a Timer with an extremely long counter period as the default prescaler is only 16-Bit and Timers are up to 32-Bit.

**PWM Mode**

Pulse width modulation mode of a timer permits to generate a signal with a frequency and a duty cycle. The frequency of the PWM-Signal can be configured with the Auto-Reload Register (TIMx\_ARR) and the duty cycle with the TIMx\_CRRx register. Calculating the values of these registers is part of [Task B](#_Task_B:_Vary).

Use Cases:

* Dimming an LED.

Figure 2: PWM Timing Diagram

* Driving a servo motor.
* Control the voltage in switched mode power supplies.

**Counter Mode**

The counter of a timer is the core of its functionality, as all timer operations are based of measuring discretized timesteps, where the number of elapsed timesteps is contained in the counter register (TIMx\_CNT) and counts with a controllable frequency if the timer is started. There are several modes in which the counter operates: Up-counting Mode, down-counting mode, and center-aligned mode (up/down counting). In this exercise we will only consider the up-counting mode.

Ein Bild, das Mond, Dunkelheit, Schwarz, Nacht enthält.

Automatisch generierte Beschreibung

Figure 3: Timer Counter Diagram

A use case would be to repeatedly perform a certain action with a certain period of time.

1. **Operation Mode: Timer with counter register**

*Describe verbally how this mode works. Then discuss the following example based on the diagram:*

The counter register (TIMx\_CNT) of a timer stores an integer number (16- or 32-Bit depending on the timer) which is incremented (in up counting mode) with the provided clock frequency. The clock frequency which triggers the count operation is previously divided by a prescaler value (TIMx\_PSC) to slow down the clock speed according to the use case. When the counter reaches the auto-reload value (content of the TIMx\_ARR register, by default 16- or 32 Bit), it restarts from 0 and generates a counter overflow event. The Update Interrupt Flag (UIF) which is set on an overflow event can then be read from the timer status register (TIMx\_SR) and must be reset via software.

*Ein Bild, das Text, Schrift, Screenshot, Electric Blue (Farbe) enthält.

Automatisch generierte Beschreibung*

Figure 4: Timer with Counter Register Example

|  |  |  |
| --- | --- | --- |
| Clock frequency | Divider (Prescaler) | 16-bit Counter Register at t = 0s: |
| 10 kHz | 250 | 65 |

*When will the overflow occur?*

Count period: The counter is incremented every 25ms.

The time it takes to count from 65 to the maximum 16-Bit value can be calculated as follows:

*What happens after this overflow?*

As the counter register overflows (reaches its maximum value) an update event is triggered, setting the Update Interrupt flag high and resets the counter to 0.

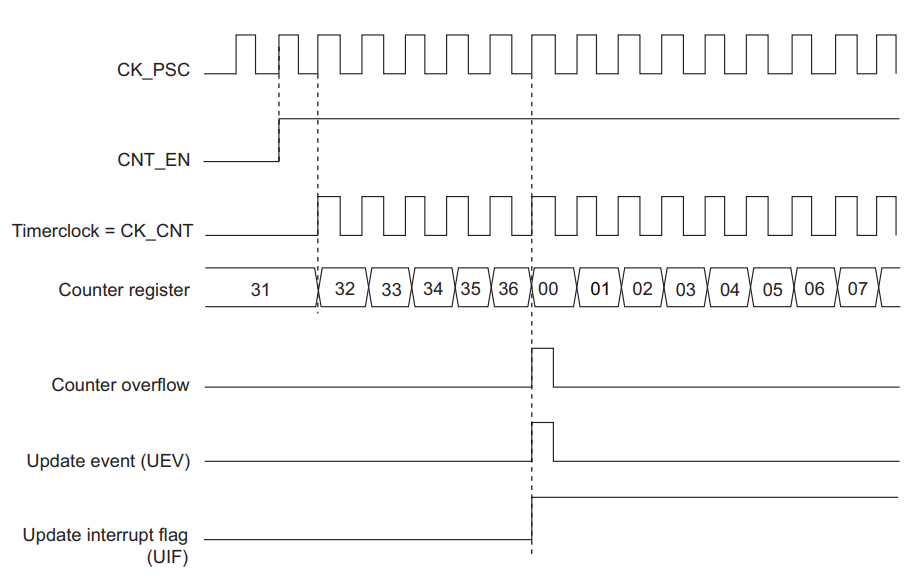


Figure 5: Counter timing diagram with PSC = 1 and ARR = 0x36

1. **Operation Mode: Timer with capture event**

*Describe verbally how this mode works. Then discuss the following example based on the diagram:*

A general purpose timer contains 4 capture and compare channels. In input capture mode the capture/compare registers (TIMx\_CCRx) are used to latch the value of the counter whenever a special input capture event occurs. A capture action can be performed by checking the Capture/Compare Interrupt Flag (CCxIF) in the timer status register (TIMx\_SR). CCxIF must then be cleared by software.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Figure 6: Timer with Capture Event Example

|  |  |  |
| --- | --- | --- |
| Clock frequency: | Divider: | Capture Event: |
| 200 MHz | 500 000 | After 2s |

*Capture the state of all registers at t = 2s when the capture event happens. What is the state of these registers?*

With a counter frequency of , the captured value of the counter register is .

*How can the board calculate the time span from this register - as it doesn’t know that 2 s have passed?*

Since the Timer Module knows at which frequency it is running, we can use backwards calculations to obtain the exact time duration.

1. **Operation Mode: Timer with compare register:**

*Describe verbally how this mode works. Then discuss the following example based on the diagram:*

The Input is captured by a catpure event in the same way as in the previous example. This time however, a second channel compares its compare register (TIMx\_CCRx) against the capture register from the first channel, which – if matches – triggers a compare action. This can be again checked with the CCxIF in the TIMx\_SR. Note that the capture register and the compare register are both called TIMx\_CCRx and the Interrupt Flag CCxIF but from different channels.

*Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung*

Figure 7: Timer with Compare Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clock frequency | Divider | Counter | Capture Register | Compare Register |
| 20 kHz | 10 000 | 0 | 0 | 10 |

*Specify all steps until the compare action is triggered, i.e., counter register, capture register and compare register - and relevant time stamps when any of these registers is updated.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Time** | **Counter** | **Capture** | **Compare** | **Actions** |
| 0 us | 0 | 0 | 10 | none |
| 1 s | 2 | 0 | 10 | none |
| 2,5 s | 5 | 5 | 10 | Capture Action |
| 3,5 s | 7 | 5 | 10 | none |
| 5 s | 10 | 10 | 10 | Capture Action  Compare Action |
| 7 s | 14 | 10 | 10 | none |
| … | … | … | 10 | … |

Table 1: Example Timestamps with Combined Capture and Compare Channels

# Task A: Blink an LED with Timer Module

In this task, you are required to toggle the LED on the Elite-Board periodically using a timer. The clock frequency of the timer module should be configured according to the requirements given below. **Requirements:**

* Use **PJ7 (SEGDP)** as digital output (LED)
* **Timer 2** module should be used for blinking the LED
* Tout should be **500 ms**.

## A.1. Calculations

To achieve a custom frequency , the frequency from the clock source can be divided with the prescaler value:

The goal is to trigger an update event every time the Auto Reload Register overflows within a specified amount of time. In this case the time is **500ms.** To calculate the time it takes the counter register to count from to the value, we simply multiply the value with the count interval:

Now, a fitting combination of and must be chosen. Since Timer 2 is a 32-Bit timer, the max value for the is high enough to use the base frequency and is chosen to be 1. The value for the can therefore easily be calculated:

|  |  |  |
| --- | --- | --- |
|  |  | formula 1 |

The value for can be set in the Clock configuration in Cube MX and is by default.

|  |  |
| --- | --- |
| Final Values | |
|  |  |
|  |  |

Table 2: Task A - Final Values from Calculations

## A.2. Implementation

**Configuration in Cube MX**

Before discussing the code implementation, timer 2 needs to be configured in Cube MX. The clock source has been set as the internal clock and the values for the and are adopted from the calculations above.

|  |  |
| --- | --- |
|  |  |

Figure 8: Task A - TIM2 Configuration in Cube MX

**Code:**

The timer can be started with the HAL provided function. The status register of the timer 2 is constantly checked if an update event has been triggered. The check can be performed by inspecting the status of the Update Interrupt Flag.

Timer 2 Status Register (TIM2->SR)

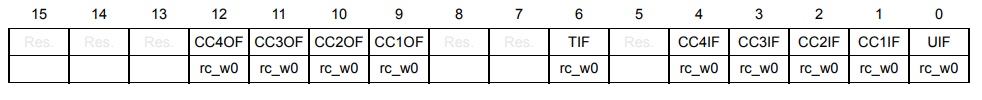


Figure 9: Timer Status Register

TIM\_SR\_UIF = 0x0001 (Bitmask for the Flag)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Flag is Set | Flag is Reset | Operation |
| TIM2->SR | 0b xxxx … xxx1 | 0b xxxx … xxx0 | & |
| TIM\_SR\_UIF | 0b 0000 … 0001 | 0b 0000 … 0001 |
| check-result | 1 | 0 | = |

Table 3: Check if a Bit is Set in Register

Because the UIF is not reset automatically, it must be manually reset in the software:

|  |  |  |
| --- | --- | --- |
| TIM2->SR | 0b xxxx … xxx1 | & |
| ~TIM\_SR\_UIF | 0b 1111 … 1110 |
| new TIM2->SR value | 0b xxxx … xxx0 | = |

Table 4: Clear Bit in Register

Of course, the decimal point LED od the segment is also toggled.



Code Segment 1: Task A - Relevant Code in main.c

## A.3. Discussion

Because this code is relatively simple, no functions or code outside of the main function is implemented. Still, the code is very easy to understand due to HAL provided macros and functions from the GPIO and TIM module.

The difference to practicum 1, where the HAL\_Delay() function was used to achieve the blinking period, is that the timer is not part of the code but a separate unit which counts independently. This is favorable because it allows other functions (such as button press-checks) to be executed without being blocked by any delay.

# Task B: Vary the Brightness of an LED

The user should be able to control the brightness of the LED by using two push buttons provided on the board. These buttons are connected to the pins PJ12 and PJ13 and labelled as BTN1 and BTN2 respectively. BTN1 button should be used to increase the brightness of the LED and BTN2 is used to decrease the brightness. The range of brightness should be between 0% and 100% and an individual step should be 10%. That means, increasing the brightness by a single step indicates a 10% increase in brightness and vice versa. The brightness should be controlled by adjusting the PWM signal which can be generated using a timer module. The configuration of the timer module and the PWM signal should be done according to the below requirements.

**Requirements:**

* Use **PJ7 (SEGDP)** as digital output.
* Use **PJ12 (BTN1)** and **PJ13 (BTN2)** for digital inputs.
* Frequency of the PWM signal should be **500 Hz**.
* Use **channel 2** of **TIM8** for generating PWM signal (**PJ6**)
* Max brightness = **100%**, Min brightness = **0%**, Step size = **10%**

## B.1. Calculations

To vary the brightness of the LED, a PWM signal is used to control the average power delivered to the LED. The PWM mode of a timer works as depicted in Figure 2.

**Calculate the frequency of the PWM-Signal:**

First, the frequency is divided by the prescaler . The prescaled timer frequency then counts to the auto reload registers value. The formula of frequency for the whole cycle time is hereby:

With the specification for and (defined in the Clock Configuration tab in CubeMX as APB Timer Clock), an arbitrary combination of the prescaler value and the Auto-Reload-Register value can be chosen. But to achieve a higher resolution for the duty cycle, the 16-Bit value should be maximized. Therefor the lowest possible prescaler value was chosen which is . Since the frequencies are given and we can freely choose the prescaler, the formula can be converted to .

**Calculate the duty cycle:**

The duty cycle is the *on*- to *cycle* time ratio. Since CCR spans over the *on* period of the counter and ARR over the whole period, the duty cycle can be calculated as follows:

To calculate the duty cycle from 0 to 100 with a step size of 10%, set the capture/compare register to

|  |  |  |
| --- | --- | --- |
|  |  | formula 2 |

where the integer number is in the interval of . This register is updated on every button push.

|  |  |
| --- | --- |
| Final Values | |
|  |  |
|  |  |

Table 5: Task B - Final Values from Calculations

## B.2. Implementation

**Configuration in Cube MX:**

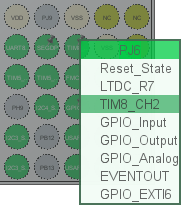
Conveniently, the *GPIO PJ6* which is wired to segment G is also internally connected to channel 2 of Timer 8. The first thing to be done, is to select it in the pinout of the microcontroller as shown in Figure 10. After selecting PWM-Generation (non-inverted with output) for channel 2, the values for the prescaler and auto reload register can be inserted from the calculations.

Figure 10: PJ6 as TIM8 CH2 in Pinout

The *Pulse* value, which determines the duty cycle in the *PWM Generation Channel 2* configuration, can be initially set to 0, because it will be dynamically adjusted to the state of the program.

|  |  |
| --- | --- |
|  |  |

Figure 11: Task B - Config for TIM 8

**Code:**

For this task, the job of the main function is to firstly initialize the two buttons used to step up and down with the duty cycle, as well as starting the PWM Output of Timer 8 channel 2. An integer variable is used to keep track of the current step.

*The functions for the button module are reused from practicum 1, where they are also documented in detail. Therefore, there won’t be any code snippets in this document but are still available in the appended code.*

In the infinite while loop, both buttons are checked for updates. With the results of these checks a sign sgn variable is computed to determine the direction of change. The conditions for this variable are listed in following table:

|  |  |  |
| --- | --- | --- |
| step\_up.has\_changed | step\_dn.has\_changed | sgn  (step\_up.has\_changed - step\_dn.has\_changed) |
| 0 | 0 | 0 |
| 0 | 1 | -1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 6: Conditions for sgn variable

This variable can then be directly added to the current step (curr\_step) and checked if it exceeds the interval from MIN\_STEP (0) to MAX\_STEP (10). In case it does, the value is capped to either MIN\_VALUE or MAX\_VALUE. Finally, the Capture/Compare Register which determines the pulse width can be updated with the new value derived from formula 2.



Code Segment 2: Task B - Relevant Code in main.c

## B.3. Results

Following Figures show results of the PWM controlled G-Segment of the display.

|  |  |
| --- | --- |
| Figure 12: Task B Result - Low Intensity (10%) | Figure 13: Task B Result - High Intensity (100%) |

## B.4. Discussion

*To emphasize previous feedback regarding the use of macros:*

The reason a macro instead of a function was implemented in the last practicum is because the functionality of this macro was needed for different data types, the issues and potential problems of macro implementations were also discussed in the protocol. The macro used:



Code Segment 3: LIMIT Macro

For example, x = LIMIT(x, max, min) can be interpreted as



for better readability. Since exactly this functionality was needed multiple times in the code, we decided to create a macro for this operation. Even though this implementation works perfectly fine for this case and will still be used in the code, a more secure version could be implemented as follows:



Code Segment 4: Secure LIMIT Macro

Implementing it as demonstrated ensures that each input parameter is only used once, and its value is copied to local parameters instead.

# Task C: Count Pulses of a Blinking LED

In this task, you are required to generate a pulse counter with Timer 3. A clock (Tout = 1 sec approx.) signal shall be generated from Timer 2, which shall be used as an internal trigger for Timer 3. Timer 3 shall count repeatedly from 0 to 9. The current count of the Timer 3 should be displayed on the Seven Segment Display of the Eliteboard.

**Requirements:**

* Use the **7-segment display** for digital output.
* **Timer 2** should provide an internal trigger; **Tout** should be **1s**.
* **Timer 3** should be used to count (**0-9**) pulses.

## C.1. Implementation

In this task, two timers have to be configured in such a way, that they are connected as shown in Figure 14. This is a case where a master timer acts as a prescaler for a slave timer.

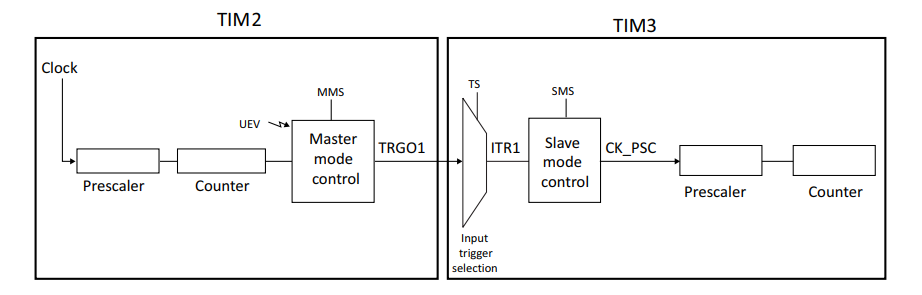


Figure 14: Task C - Timer Interconnection Diagram

**Configuration in Cube MX:**

*Master Timer 2 Configuration:*

|  |  |
| --- | --- |
|  |  |

Figure 15: Task C - Master Timer Configuration

The ARR value is calculated in the same way as in formula 1 but with . A prescaler of 0 means, that no prescaler is used and is equal to a prescaler of 1. Since the Trigger output (TRGO) causes the internal trigger to fire, we want it to respond to an overflow update event.

*Slave Timer 3 Configuration:*

|  |  |
| --- | --- |
|  |  |

Figure 16: Task C - Slave Timer Configuration

Timer 3 is configured in slave mode. Instead of the internal clock, external clock mode 1 is now selected because as depicted in Figure 17 from the datasheet [RM0433, p. 1646], the Internal Trigger (ITRx) is connected to this clock mode.

Ein Bild, das Text, Diagramm, Plan, Reihe enthält.

Automatisch generierte Beschreibung

Figure 17: External Clock Connection Diagram

To know which internal trigger is connected to which master timer TRGO, Table 7 can be found in the datasheet [RM0433, p. 1682]. With timer 3 as the slave timer and timer 2 as the master timer, the internal trigger is ITR1.

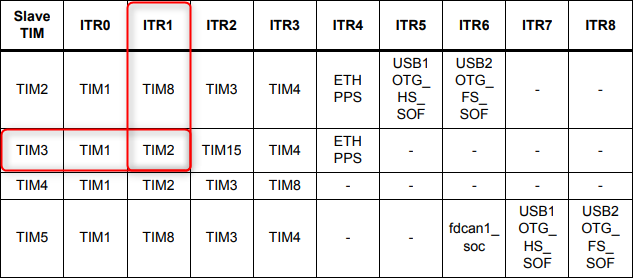


Table 7: TIMx Internal Trigger Connection

Finally, since we want the slave timer to count from 0 to 9, the ARR is set correspondingly to 9.

**Code:**

In the main method for this task, the previously configured timers are started, and a segment display is instantiated. It is checked in the infinite while loop, if the counter register changed its value compared to the currently displayed number. If so, the segment display is directly written with the counter register value of the slave timer 3, where seg.curr\_num is updated as well.



Code Segment 5: Task C - Relevant Code in main.c

The write function for the segment display limits the value to the set values (0 and 9) and writes the number to the Output Data Register. MODIFY\_REG is a macro provided by the Hardware Abstraction Layer Library.



Code Segment 6: Segment Write Function

* seg->GPIOx is the port the segment display is connected to (Port J)
* seg->pin\_mask is the or-ation of every SEGx\_Pin (x … A-G)
* seg->segments[] is an array containing the pin mask of each number where the index is that corresponding number (0-9).

## C.2. Discussion

We noticed that the difficulty in this task was not the actual coding but finding relevant information in the datasheet. Additionally, the ability to reuse a lot of functions from practicum 1 made it easier to focus on the timer implementation, which was very convenient.

# Index

CCxIF *Capture/Compare x Interrupt Flag (x = channel)*

CK\_INT *Internal Clock*

ETR *External Trigger*

GPIO *General Purpose Input/Output*

HAL *Hardware Abstraction Layer*

ITRx *Internal Trigger (Nr x)*

LED *Light Emitting Diode*

TIM *Timer*

TIMx\_ARR *Timer Auto-Reload Register*

TIMx\_CNT *Timer Counter Register*

TIMx\_CRRx *Capture Compare Register*

TIMx\_PSC *Prescaler Register*

TIMx\_SR *Timer Status Register*

TIx *External Input Pin (Nr x)*

TRGO *Trigger Output*

UEV *Update Event*

UIF *Update Interrupt Flag*

## Figures

[Figure 1: Master/Slave Timer Example 3](#_Toc166212420)

[Figure 2: PWM Timing Diagram 3](#_Toc166212421)

[Figure 3: Timer Counter Diagram 4](#_Toc166212422)

[Figure 4: Timer with Counter Register Example 4](#_Toc166212423)

[Figure 5: Counter timing diagram with PSC = 1 and ARR = 0x36 5](#_Toc166212424)

[Figure 6: Timer with Capture Event Example 5](#_Toc166212425)

[Figure 7: Timer with Compare Register 6](#_Toc166212426)

[Figure 8: Task A - TIM2 Configuration in Cube MX 8](#_Toc166212427)

[Figure 9: Timer Status Register 8](#_Toc166212428)

[Figure 10: PJ6 as TIM8 CH2 in Pinout 12](#_Toc166212429)

[Figure 11: Task B - Config for TIM 8 12](#_Toc166212430)

[Figure 12: Task B Result - Low Intensity (10%) 14](#_Toc166212431)

[Figure 13: Task B Result - High Intensity (100%) 14](#_Toc166212432)

[Figure 14: Task C - Timer Interconnection Diagram 16](#_Toc166212433)

[Figure 15: Task C - Master Timer Configuration 16](#_Toc166212434)

[Figure 16: Task C - Slave Timer Configuration 16](#_Toc166212435)

[Figure 17: External Clock Connection Diagram 17](#_Toc166212436)

## Code Segments

[Code Segment 1: Task A - Relevant Code in main.c 9](#_Toc166212437)

[Code Segment 2: Task B - Relevant Code in main.c 13](#_Toc166212438)

[Code Segment 3: LIMIT Macro 14](#_Toc166212439)

[Code Segment 4: Secure LIMIT Macro 14](#_Toc166212440)

[Code Segment 5: Task C - Relevant Code in main.c 18](#_Toc166212441)

[Code Segment 6: Segment Write Function 18](#_Toc166212442)

## Tables

[Table 1: Example Timestamps with Combined Capture and Compare Channels 6](#_Toc166212443)

[Table 2: Task A - Final Values from Calculations 7](#_Toc166212444)

[Table 3: Check if a Bit is Set in Register 8](#_Toc166212445)

[Table 4: Clear Bit in Register 8](#_Toc166212446)

[Table 5: Task B - Final Values from Calculations 11](#_Toc166212447)

[Table 6: Conditions for sgn variable 13](#_Toc166212448)

[Table 7: TIMx Internal Trigger Connection 17](#_Toc166212449)