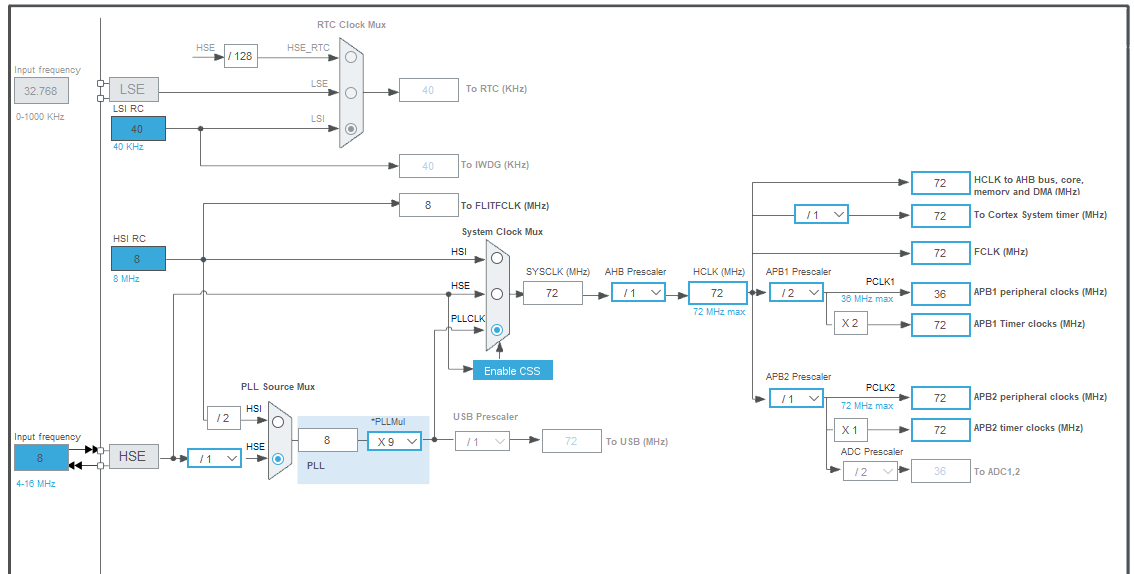
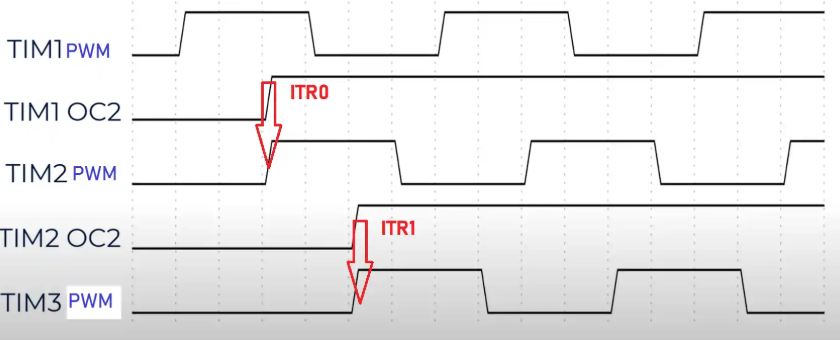
# Introduction:

In the previous project, the Trigger mode was utilized to initiate the counter of the slave timer. Under this configuration, the slave counter commenced operation only upon the overflow of the master timer's counter. However, consider a scenario where the slave counter must begin operation when the master counter reaches a specified value. This represents a fundamental requirement for generating 3-phase or 2-phase PWM signals, which cannot be accomplished using the previously implemented approach. The initiation of the slave counter is regulated by establishing a specific threshold value for the master counter, enabling precise control over its activation.

# Clock configuration:



Here both the APB Timer clocks are running at 72 MHz. We will use the timers 1, 2 and 3 to create the 3 PWM signals. Here TIM1 is connected to the APB2 bus and TIM2 and 3 are connected to the APB1 bus.



the TIM2 as a slave can be controlled by the master TIM1 using the ITR0 signal. Similarly, the TIM3 can be controlled by the TIM2 using the ITR1 signal. these signals are used to trigger the counter of the slave Timers, when the master counter reaches a predefined value.

* Firstly, the TIM1 will generate a PWM signal on channel 1
* When the counter 1 will reach 33% of the ARR value, TIM1 Output Compare on channel 2 will go high.
* This will trigger the ITR0 signal, and TIM2’s counter will start at that moment.
* Similarly, when the counter 2 will reach 33% of the ARR value, TIM2 Output compare on channel 2 will go high and it will trigger the ITR1 signal.
* TIM3 counter will start at this moment.

# Prescaler and ARR:

## Prescaler:

Prescaler is a divider which divides frequency of clock of micro by a fix number to the timer frequency before it applied to the timer.

The prescaler values, referred to as prescaler, that may be configured might be limited to a few fixed values (powers of 2), or they may be any integer value from 1 to 2^P, where P is the number of prescaler bits.

The purpose of the prescaler is to allow the timer to be clocked at the rate a user desires. For shorter (8 and 16-bit) timers, there will often be a tradeoff between resolution (high resolution requires a high clock rate) and range (high clock rates cause the timer to overflow more quickly).

Prescaler is based on the bit value so the number range of it starts with 0 and it ends based on the bit value.

In the micro controller we used stm32f103c8t6 there is a 16bit prescaler for timer so the range of prescaler is [0-65535] as the prescaler starts with 0 so we set a counter to count from 0 to a given value. The counter includes both the starting value and the ending value. This means there are actually (value+1) counts in total.

In formula we have so we have to set perscaler-1 when we want the actual value.

## Counter period (ARR):

The counter period or ARR refers to the value that determines the duration of the timers counting cycle.it defines the maximum count value that the timer will reach before it resets to 0 and starts counting again.

ARR is based on the bit value so the number range of it starts with 0 and it ends based on the bit value.

In the micro controller we used stm32f103c8t6 there is a 16bit counter period for timer so the range of counter period is [0-65535] as the counter period starts with 0 so we set a counter to count from 0 to a given value. The counter includes both the starting value and the ending value. This means there are actually (value+1) counts in total.

In formula we have so we should set the ARR-1 when we want the actual value.

The information of project

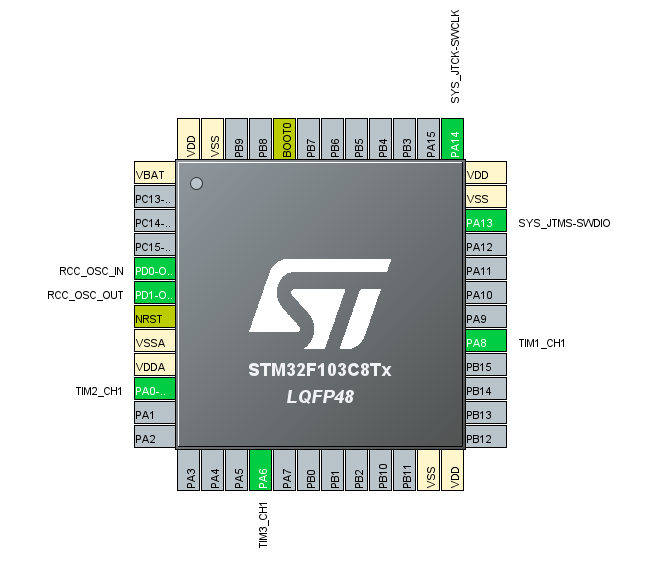
Frequency =10khz

So if we set ARR=100 and prescaler= 72 then frequency set 10 khz

Duty cycle= 70%.

Since both the APB Timer clocks are running at the same 72MHz frequency, I have used the similar configuration for all three timers.  
Here the timers will run at 100 Hz frequency with the PWM duty as 70%.

# Pinout:

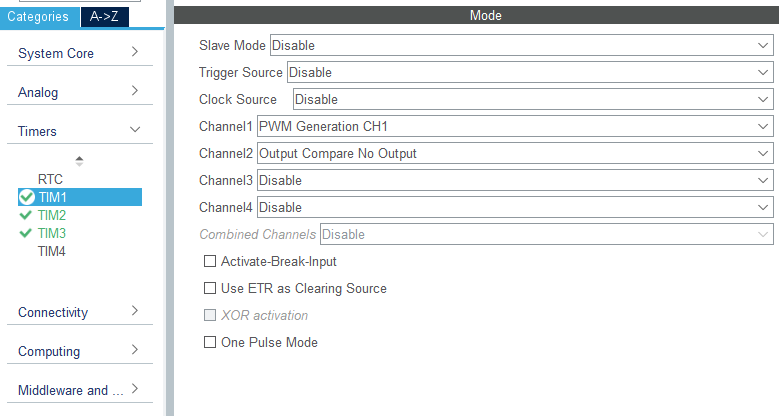


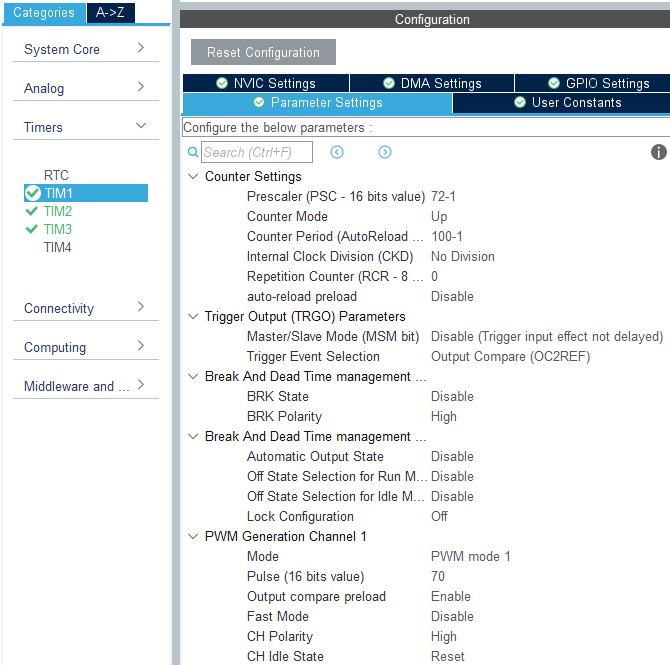
These 3 pins are basically the PWM pins from each timer.

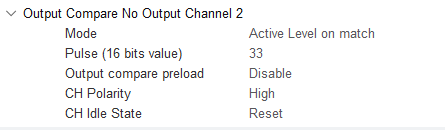
# Timer configs:

# Timer1:

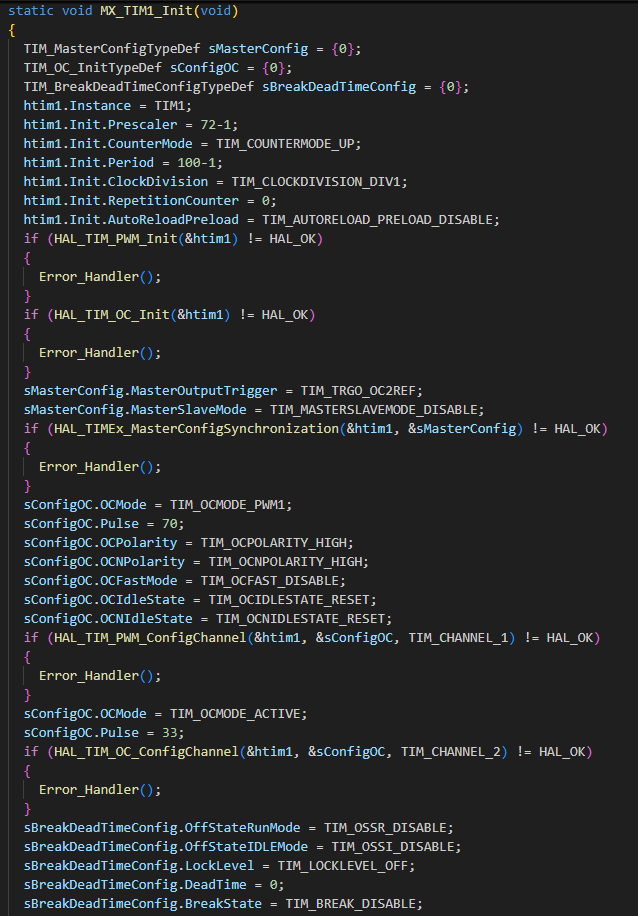
* The TIM1 is the master for TIM2, so there is no slave mode for it.
* The TIM1 has the channel 1 set as the PWM output, and channel 2 as the Output compare.
* The output compare (channel2) signal is used as the trigger source for the next Timer in line.
* The prescaler is set to 71 (72-1=71) because it is based on the binary value and starts from “0” bit.
* ARR is set to 99 (100-1=99) because it is also based on the binary value and starts from “0” bit.
* The pulse value for the pwm generation channel 1 is set at 70 which mentions to the duty cycle.
* The pulse value (CCR value) for the output compare channel 2 is set at 33. This is basically the 33% of the ARR value, which is 99.







# This configs generates this code function:



### Code Breakdown:

1. **Function Declaration**

static void MX\_TIM1\_Init(void)

This declares a function to initialize the TIM1 timer. The static keyword indicates that this function is only accessible within this file.

1. **Variable Initialization**

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

TIM\_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

These structures are initialized to hold configuration details:

sMasterConfig for timer master configuration.

sConfigOC for output compare (OC) configuration.

sBreakDeadTimeConfig for break/dead-time configurations. Dead-time refers to an intentional delay introduced between switching signals for complementary outputs (High-Side and Low-Side switches in a half-bridge or full-bridge circuit). This delay prevents both switches from being ON simultaneously, which would cause a short circuit through the power supply

1. **Instance Configuration**

htim1.Instance = TIM1;

Assigns the TIM1 hardware instance to the htim1 structure, which represents the timer being configured.

1. **Timer Configuration**

* htim1.Init.Prescaler = 72-1;

Sets the prescaler to 72. This affects the timer's frequency.

* htim1.Init.CounterMode = TIM\_COUNTERMODE\_UP;

Configures the timer to count upwards.

* htim1.Init.Period = 100-1;

Sets the timer's period, defining when it will reset and trigger events (ARR).

* htim1.Init.RepetitionCounter = 0;

Defines the repetition counter (used in PWM modes). Here, it’s disabled.

1. **Initialize TIM1 in PWM Mode**

if (HAL\_TIM\_PWM\_Init(&htim1) != HAL\_OK)

{

Error\_Handler();

}

Initializes the TIM1 timer in PWM (Pulse Width Modulation) mode. If it fails, the Error\_Handler function is called.

1. **Initialize Output Compare**

if (HAL\_TIM\_OC\_Init(&htim1) != HAL\_OK)

{

Error\_Handler();

}

Initializes TIM1 for Output Compare mode. Any errors are handled.

1. **Master Configuration**

* sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_OC2REF;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

Configures the master timer output trigger (OC2REF) and disables master-slave mode. OC2REF refers to the Output Compare 2 Reference signal, which is an internal signal generated by the timer in the microcontroller. It represents the output of the timer channel 2's compare function and is based on the value set in the Output Compare Register for Channel 2.

* if (HAL\_TIMEx\_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

Synchronizes the master configuration. Errors are handled as usual.

1. **PWM Channel Configuration**

* sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 70;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCNPolarity = TIM\_OCNPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

sConfigOC.OCIdleState = TIM\_OCIDLESTATE\_RESET;

sConfigOC.OCNIdleState = TIM\_OCNIDLESTATE\_RESET;

This sets the PWM mode (PWM1), output pulse width=duty cycle (70), polarity, fast mode, and idle state configurations.

 sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

* Sets the polarity of the output signal for the primary channel. TIM\_OCPOLARITY\_HIGH means the active level is high.

 sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

* Disables fast mode for the Output Compare. Fast mode allows immediate output changes upon receiving a trigger, but disabling it ensures standard timing behavior and stability for PWM signals.

 sConfigOC.OCIdleState = TIM\_OCIDLESTATE\_RESET;

* Specifies the idle state of the primary output when the Timer is disabled. TIM\_OCIDLESTATE\_RESET means the output returns to a low state.
* if (HAL\_TIM\_PWM\_ConfigChannel(&htim1, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

Applies these configurations to channel 1. Errors are checked.

1. **Active Output Compare Configuration**

sConfigOC.OCMode = TIM\_OCMODE\_ACTIVE;

sConfigOC.Pulse = 33;

This changes the mode to ACTIVE with a 33 pulse for channel 2.

if (HAL\_TIM\_OC\_ConfigChannel(&htim1, &sConfigOC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

Configures channel 2 with the above settings.

1. **Break and Dead-Time Configuration**

sBreakDeadTimeConfig.OffStateRunMode = TIM\_OSSR\_DISABLE;

sBreakDeadTimeConfig.OffStateIDLEMode = TIM\_OSSI\_DISABLE;

sBreakDeadTimeConfig.LockLevel = TIM\_LOCKLEVEL\_OFF;

sBreakDeadTimeConfig.DeadTime = 0;

sBreakDeadTimeConfig.BreakState = TIM\_BREAK\_DISABLE;

sBreakDeadTimeConfig.BreakPolarity = TIM\_BREAKPOLARITY\_HIGH;

sBreakDeadTimeConfig.AutomaticOutput = TIM\_AUTOMATICOUTPUT\_DISABLE;

These settings manage the timer's break features and dead-time. Dead-time is disabled, and no automatic output is configured.

if (HAL\_TIMEx\_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL\_OK)

{

Error\_Handler();

}

Applies the break/dead-time settings. Errors are checked.

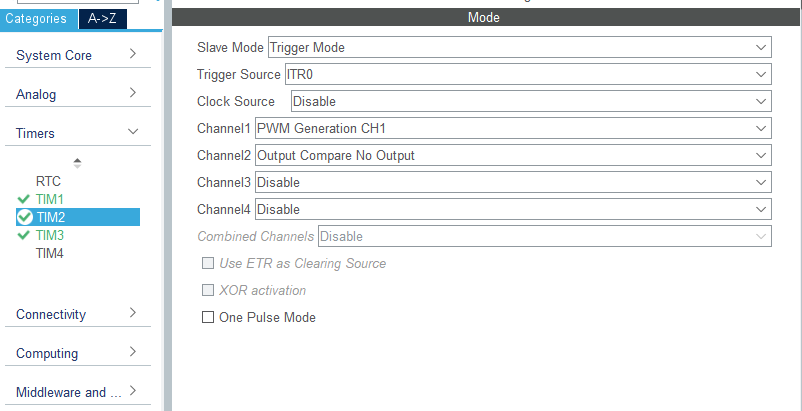
1. **Post-Initialization**

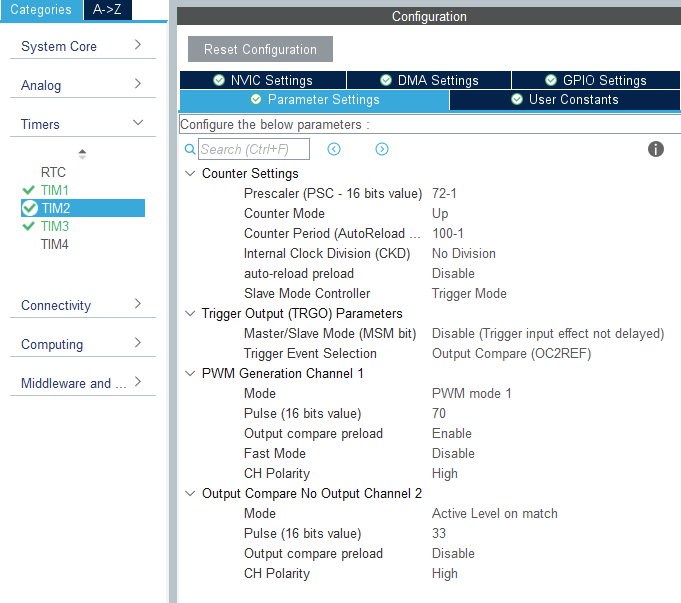
HAL\_TIM\_MspPostInit(&htim1);

Completes any low-level configuration required for TIM1 after initialization. It performs any additional, low-level hardware configuration necessary after the timer has been initialized.

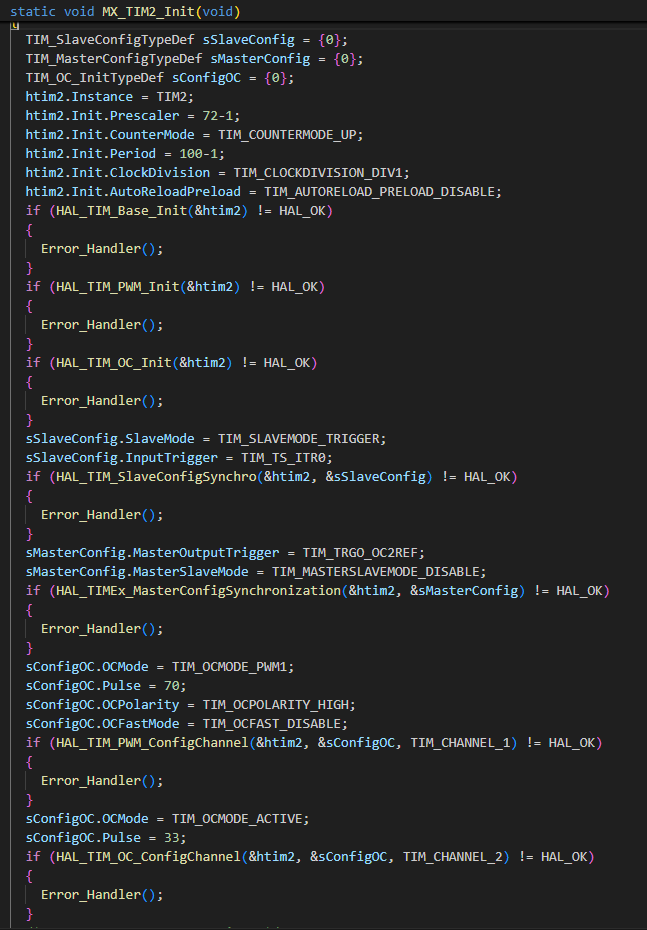
# Timer2:

* TIM2 is the slave for TIM1, which can be triggered by the ITR0 signal.
* The TIM2 is also the master for TIM3.
* TIM2 has the channel 1 set as the PWM output, and channel 2 as the Output compare.
* The output compare signal is used as the trigger source (ITR0) for the next Timer (TIM3).
* The prescaler is set to 71 (72-1=71) because it is based on the binary value and starts from “0” bit.
* ARR is set to 99 (100-1=99) because it is also based on the binary value and starts from “0” bit.
* The pulse value for the pwm generation channel 1 is set at 70 which mentions to the duty cycle.
* The pulse value (CCR value) for the output compare channel 2 is set at 33. This is basically the 33% of the ARR value, which is 99.





# This configs generates this code function:



### Code Breakdown:

1. **Variable Initialization**

TIM\_SlaveConfigTypeDef sSlaveConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

These structures are initialized to hold various timer configurations:

* + sSlaveConfig: For setting slave mode configuration.
  + sMasterConfig: For setting master timer configuration.
  + sConfigOC: For Output Compare (OC) configuration.

1. **Timer Instance Assignment**

htim2.Instance = TIM2;

This specifies that the configuration will apply to **TIM2** hardware timer.

1. **Timer Initialization Settings**

htim2.Init.Prescaler = 72-1;

Sets the prescaler to 72. This affects the timer's frequency.

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

Configures the timer to count upwards.

htim2.Init.Period = 100-1;

Defines the timer's maximum count value. When this is reached, the timer resets and starts over (ARR).

htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

Disables preload for the auto-reload register.

1. **Base Timer Initialization**

if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

Initializes TIM2 in basic mode. Errors during initialization trigger the Error\_Handler function.

1. **PWM Initialization**

if (HAL\_TIM\_PWM\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

Configures TIM2 for **PWM (Pulse Width Modulation)** mode.

1. **Output Compare Initialization**

if (HAL\_TIM\_OC\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

Initializes TIM2 for **Output Compare** mode.

1. **Slave Configuration**

sSlaveConfig.SlaveMode = TIM\_SLAVEMODE\_TRIGGER;

sSlaveConfig.InputTrigger = TIM\_TS\_ITR0;

* Sets TIM2 to **trigger mode** to function as a slave timer.
* Configures ITR0 as the input trigger source. ITR0 is a predefined signal available in STM32 timers. It is part of the **inter-timer trigger system**, allowing timers to synchronize with one another without needing external hardware signals.
* STM32 timers are interconnected, and internal trigger sources (like ITR0, ITR1, ITR2, etc.) allow one timer (the **master**) to act as the source of synchronization signals for another timer (the **slave**).

if (HAL\_TIM\_SlaveConfigSynchro(&htim2, &sSlaveConfig) != HAL\_OK)

{

Error\_Handler();

}

Applies the slave configuration and checks for errors.

1. **Master Configuration**

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_OC2REF;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

* + Sets the **output trigger** as OC2REF for synchronization.
  + Disables master-slave mode.

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

Synchronizes the master timer configuration.

1. **PWM Channel 1 Configuration**

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 70;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

* + Configures PWM mode (PWM1), with a pulse width of 70 clock cycles.
  + Sets high polarity for the output signal.

if (HAL\_TIM\_PWM\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

Applies these settings to channel 1.

1. **Output Compare Channel 2 Configuration**

sConfigOC.OCMode = TIM\_OCMODE\_ACTIVE;

sConfigOC.Pulse = 33;

Configures Output Compare mode (ACTIVE) for channel 2, with a pulse width of 33 clock cycles.

if (HAL\_TIM\_OC\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

Configures channel 2 with the above settings.

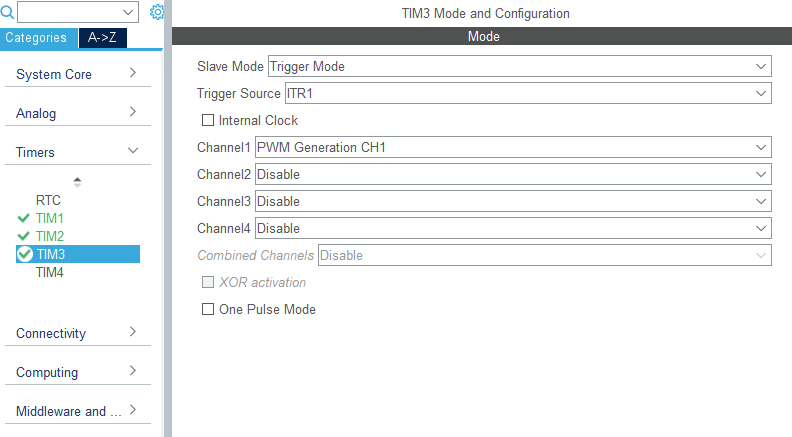
1. **Post-Initialization**

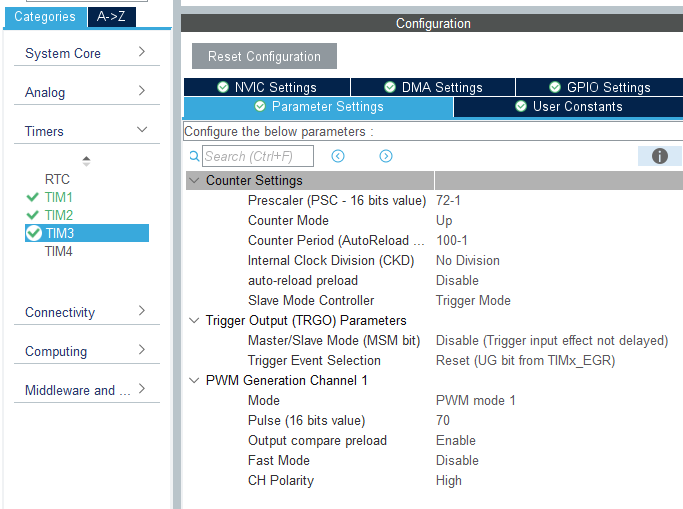
HAL\_TIM\_MspPostInit(&htim2);

Performs additional configuration steps required after initialization.

# Timer3:

* TIM3 is the slave for TIM2, which can be triggered by the ITR1 signal.
* The TIM3 is not master for the other timers.
* TIM3 has only the channel 1 set as the PWM output.
* The prescaler is set to 71 (72-1=71) because it is based on the binary value and starts from “0” bit.
* ARR is set to 99 (100-1=99) because it is also based on the binary value and starts from “0” bit.
* The pulse value for the pwm generation channel 1 is set at 70 which mentions to the duty cycle.







### Code Breakdown:

1. **Timer Instance Assignment**

htim3.Instance = TIM3;

* Specifies that the configuration applies to the **TIM3** hardware timer.
* The other steps are just like TIMER1 and TIMER2.

1. **Slave Configuration**

sSlaveConfig.SlaveMode = TIM\_SLAVEMODE\_TRIGGER;

sSlaveConfig.InputTrigger = TIM\_TS\_ITR1;

* + Configures TIM3 to function as a **slave timer** in **trigger mode**.
  + Specifies ITR1 as the **input trigger source**, potentially linking it to another timer or external event.

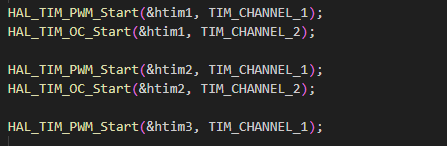
1. **Master Configuration**

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

* + Configures the **master output trigger** as TIM\_TRGO\_RESET. Due to the use of multiple timers in a **master-slave configuration**, TIM\_TRGO\_RESET ensures that the **slave timer counter starts at zero** after synchronization. it ensures peripheral devices are reset before new operations begin. This is useful for coordinating the operation of timers so that they are perfectly aligned. If no pulse generation is required, TIM\_TRGO\_RESET is used simply to synchronize or reset devices without outputting additional signals.
  + Disables **master-slave mode**.

# Main code:



HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_1): This starts the PWM output on Channel 1 of timer instance htim1.

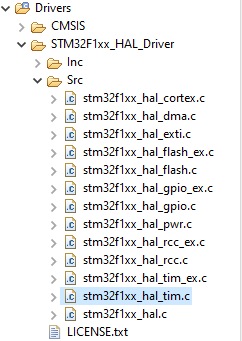
HAL\_TIM\_OC\_Start(&htim1, TIM\_CHANNEL\_2): This starts the Output Compare mode on Channel 2 of the same timer instance htim1.

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_1): This starts the PWM output on Channel 1 of timer instance htim2.

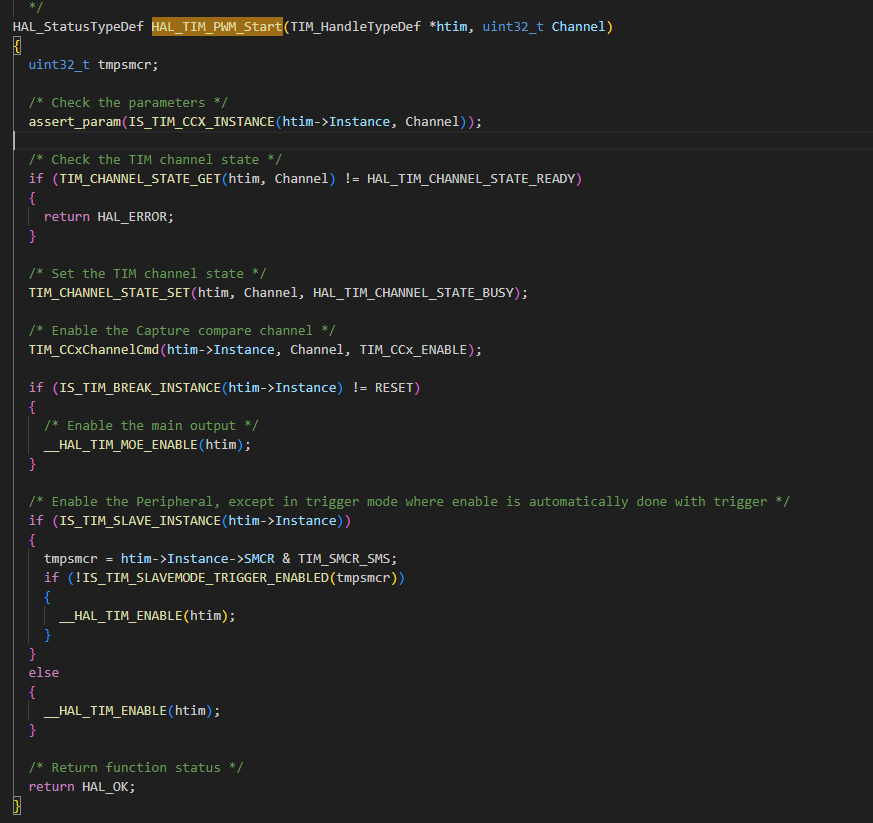
HAL\_TIM\_OC\_Start(&htim2, TIM\_CHANNEL\_2): This starts the Output Compare mode on Channel 2 of the same timer instance htim2.

HAL\_TIM\_PWM\_Start(&htim3, TIM\_CHANNEL\_1): This starts the PWM output on Channel 1 of timer instance htim3.

We can copy HAL\_TIM\_PWM\_Start function from stm32f1xx\_hal\_tim.c driver:



# function prototype:



# Code breakdown:

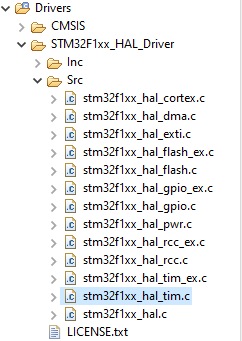
1. **Parameters**:
   * assert\_param(IS\_TIM\_CCX\_INSTANCE(htim->Instance, Channel))
   * The function takes two parameters:
     + htim: A pointer to a TIM\_HandleTypeDef structure, which holds the configuration information for the TIM peripheral.
     + Channel: The specific TIM channel to be enabled.
2. **Channel State Verification**:
   * TIM\_CHANNEL\_STATE\_GET(htim, Channel)
   * The function checks the current state of the specified TIM channel
   * If the channel is not in the HAL\_TIM\_CHANNEL\_STATE\_READY (Ready state), the function returns HAL\_ERROR.
3. **Set Channel State**:
   * TIM\_CHANNEL\_STATE\_SET(htim, Channel, HAL\_TIM\_CHANNEL\_STATE\_BUSY)
   * If the channel is ready, it changes the state to (Busy) HAL\_TIM\_CHANNEL\_STATE\_BUSY
4. **Enable Compare Channel**:
   * TIM\_CCxChannelCmd(htim->Instance, Channel, TIM\_CCx\_ENABLE)
   * It enables the compare channel.
5. **Enable Peripheral**:
   * For TIM instances that support slave mode, it checks the slave mode configuration. If the peripheral is not in trigger mode, it enables the TIM peripheral using \_\_HAL\_TIM\_ENABLE(htim).
   * Otherwise, it directly enables the TIM peripheral.

# Main code:

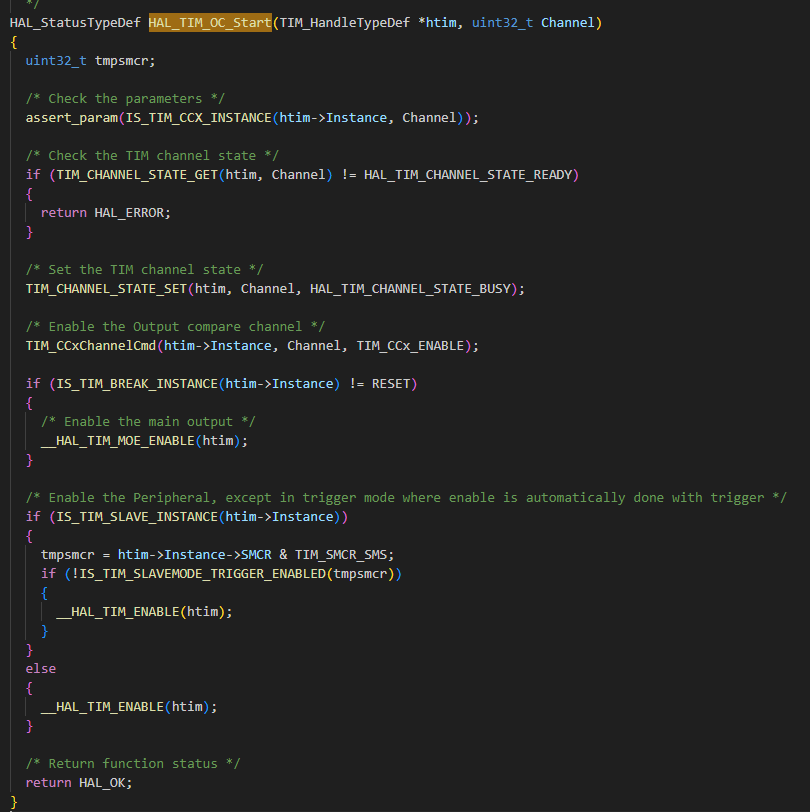
The HAL\_TIM\_OC\_Start function is used to start the Output Compare (OC) mode on a specified TIM channel. This is typically employed when you need to generate precise timing events or control output signals based on the timer's counter value.



We can copy it from stm32f1xx\_hal\_tim.c driver:



# function prototype:



# Code breakdown:

1. **Parameters**:

* assert\_param(IS\_TIM\_CCX\_INSTANCE(htim->Instance, Channel))
* The function accepts two parameters:
  + htim: a pointer to a TIM\_HandleTypeDef structure that contains the configuration information for the TIM peripheral. We want to use timer1, timer2 and timer3. So we specify htim1, htim2, htim3 to be utilized.
  + Channel: the TIM channel to be enabled.

1. **Channel State Verification**:

* TIM\_CHANNEL\_STATE\_GET(htim, Channel)
* The function checks the current state of the TIM channel
* If the channel is not in the HAL\_TIM\_CHANNEL\_STATE\_READY state, the function returns HAL\_ERROR.

1. **Set Channel State**:

* TIM\_CHANNEL\_STATE\_SET(htim, Channel, HAL\_TIM\_CHANNEL\_STATE\_BUSY)
* If the channel is ready, it sets the channel state to HAL\_TIM\_CHANNEL\_STATE\_BUSY.

1. **Enable Output Compare Channel**:

* TIM\_CCxChannelCmd(htim->Instance, Channel, TIM\_CCx\_ENABLE)
* It enables the output compare channel by calling.

1. **Enable Peripheral**:

* For TIM instances that support slave mode, it checks the slave mode configuration. If the peripheral is not in trigger mode, it enables the TIM peripheral using \_\_HAL\_TIM\_ENABLE(htim).
* Otherwise, it directly enables the TIM peripheral.

# Result:

We know the time period of the PWM signal is 10ms, so the PWM2 starts at 3.3ms. 33% of 10ms. Similarly, the PWM3 starts at 6.6ms which is 66% of 10ms.  
Basically, all three signals have 120 degrees phase difference between them.

