AN4776
Application note

How to use general-purpose timer peripheral on STM32 MCUs

Introduction

The timer peripheral is part of the essential set of peripherals embedded in all the STM32 microcontrollers. The number of timer peripherals and their respective features differ from one STM32 microcontroller family to another, but they all share some common features and operating modes.

The STM32 timer peripheral was conceived to be the keystone peripheral for a large number of applications: from motor-control applications to periodic-events generation applications. The specifications on the timer peripheral available in all STM32 reference manuals are very wide due to its versatility.

The purpose of this application note is to provide a simple and clear description of the basic features and operating modes of the STM32 general-purpose timer peripherals. This document complements the specifications of the STM32 timer peripherals available on their reference manuals.

The document is divided in two main parts:

- The first section presents the basic features of the STM32 timers in a simple way and describes some specific features that are commonly used within timer-peripheral-based applications.
- The following sections are dedicated to describe a particular use-case of an STM32 timer peripheral. These sections provide a deep description of the main STM32 timer features used to build the example application. They also describe the architecture of the used source code.

The objective of this application note is to present in a generic and simple way some use-cases of the STM32 timer peripherals, and it does not cover use-cases like motor control applications due to their complexity.

Contents

1 Basic operating modes of STM32 general-purpose timers 6

1.1 Introduction 6

1.2 STM32 timer peripheral tear-down 6

1.2.1 The master/slave controller unit 9

1.2.2 The time-base unit 9

1.2.3 The timer-channels unit 10

1.2.4 The Break feature unit 12

1.3 STM32 timer peripheral basic operating modes 12

1.3.1 Timer time-base configuration 12

1.3.2 Timer channel-configuration in input mode 13

1.3.3 Timer channel-configuration in output mode 13

1.4 STM32 timer peripheral advanced features 14

1.4.1 Filtering stage 14

1.4.2 The preload feature of the timer registers 15

2 Timer clocking using external clock-source 18

2.1 Overview 18

2.2 Synchronization block 19

2.3 The external clock-source mode 1 20

2.4 The external clock-source mode 2 22

2.5 External clock-source mode 1 versus mode 2 24

2.6 Application: timer clocking using external clock-source on ETR timer input 25

2.7 Firmware overview 33

3 N-pulse waveform generation using one-pulse mode 35

3.1 Overview 35

3.2 Application: N-pulse waveform generation using one-pulse mode 36

3.3 Firmware overview 38

4 Cycle-by-cycle regulation using break input 39

4.1 Overview 39

4.2 Break input versus OCxRef-clear utilization 39

4.3 Application: cycle-by-cycle regulation using the Break feature 41
4.4 Firmware overview 45

5 Arbitrary waveform generation using timer DMA-burst feature 47
5.1 STM32 DMA-burst feature overview 47
5.2 Timer DMA-burst feature 47
5.3 Application example: arbitrary waveform generation using timer DMA-burst feature 51

6 N-pulse waveform generation using timer synchronization 59
6.1 Timer synchronization overview 59
6.2 N-pulse waveform generation application example - part 1 61
6.3 N-pulse waveform generation application example - part2 67
6.3.1 Clock configuration 69

7 Revision history 75

List of tables

Table 1. Document revision history 75

List of figures

Figure 1. TIM1 timer-peripheral block diagram . . . 8
Figure 2. Relevant bloc diagram for the timer channel when configured as output . . . 10
Figure 3. Relevant bloc diagram for the timer channel when configured as input . . . 12
Figure 4. Input signal filtering (ETF [3:0]= 0100) : FSAMPLING = FDTS/2, N=6. . . . 15
Figure 5. Preload mechanism for timer channel register - disabled. . . . 16
Figure 6. Preload mechanism for timer channel register - enabled . . . 16
Figure 7. Synchronizing an STM32 timer by an external clock-signal . . . 18
Figure 8. Clock path for external clock-source modes . . . 19
Figure 9. Synchronization block. . . . 20
Figure 10. Timer counter increment (external clock mode 1). . . . 22
Figure 11. Timer counter increment (external clock mode 2) . . . 24
Figure 12. Synoptic of a frequency meters . . . 26
Figure 13. Frequency meter architecture clocked by the internal HSI oscillator . . . 28
Figure 14. Frequency meter architecture clocked by the external clock-source mode 2. . . . 29
Figure 15. Timing diagram of input capture . . . 30
Figure 16. The PPM resulting of the internal source clock . . . 32
Figure 17. The PPM resulting of the external source clock . . . 33
Figure 18. Project organization . . . 34
Figure 19. Example of one-pulse mode . . . 36
Figure 20. Architecture example . . . 37
Figure 21. Firmware source code organization . . . 38
Figure 22. Timing of clearing TIMx OCxREF . . . 40
Figure 23. Timing of Break function . . . 41
Figure 24. Timing of cycle-by-cycle regulation. . . . 42
Figure 25. Cycle-by-cycle regulation architecture . . . 43
Figure 26. Oscilloscope screen-shot for the obtained waveform . . . 44
Figure 27. Project organization . . . 46
Figure 28. Configuration for a timer DMA-burst transfer sequence . . . 49
Figure 29. Synoptic schema of arbitrary waveform generation using DMA-burst . . . 51
Figure 30. Arbitrary waveform generator application: targeted waveform . . . 52
Figure 31. Waveform generation data pattern stored in microcontroller memory . . . 54
Figure 32. Block diagram of arbitrary waveform generation example . . . 55
Figure 33. Arbitrary signal generation on channel1 of TIM1 timer . . . 56
Figure 34. Periodic N-pulses generation block diagram . . . 62
Figure 35. Output waveform target of periodic N-pulses generation example . . . 62
Figure 36. Periodic N-pulses generation synoptic schema . . . 63
Figure 37. Timing diagram of periodic N-pulses generation example . . . 64
Figure 38. Synoptic schema of two complementary N pulses waveform generation example . . . 67
Figure 39. Output of two N-pulses complimentary waveforms generation example . . . 68
Figure 40. Timing diagram of complimentary N-pulses waveforms generation with similar final state 70

1 Basic operating modes of STM32 general-purpose timers

1.1 Introduction

All STM32 microcontrollers embed at least one timer peripheral, and some devices embed more than one type of timer peripherals. This document covers the general-purpose timers. The general-purpose timers can be recognized from other types of STM32 timer peripherals by their name.

Within the STM32 microcontrollers documentation, a general-purpose timer peripheral is always named “TIMx timer”, where “x” can be any number and it does not reflect the number of timer peripherals embedded by a given microcontroller. For example, the STM32F100 microcontrollers embed a timer peripheral named TIM17, but the total number of timer peripherals embedded by these microcontrollers is less than 17.

In general, across the STM32 microcontrollers families, the timer peripherals that have the same name also have the same features set, but there are a few exceptions. For example, the TIM1 timer peripheral is shared across the STM32F1 series, STM32F2 series and STM32F4 series, but for the specific case of STM32F30x microcontrollers family, the TIM1 timer peripheral features a bit richer features set than the TIM1 present in the other families.

The general-purpose timers embedded by the STM32 microcontrollers share the same backbone structure; they differ only on the level of features embedded by a given timer peripheral. The level of features integration for a given timer peripheral is decided based on the applications field that it targets.

The timer peripherals can be classified as:

- Advanced-configuration timers like TIM1 and TIM8 among others.
- General-purpose configuration timers like TIM2 and TIM3 among others
- Lite-configuration timers like TIM9, TIM10, TIM12 and TIM16 among others
- Basic-configuration timers like TIM6 and TIM7 among others.

The STM32 microcontrollers are based on the Arm®(a) cores.

The application note *Introduction to timers for STM32 MCUs* (AN4013) presents a detailed overview on the STM32 timer peripherals across the different STM32 microcontroller families.

1.2 STM32 timer peripheral tear-down

All the STM32 general-purpose timer peripherals share the same backbone structure. This section tears down the advanced configuration TIM1 timer peripheral, which is the timer peripheral with the most features.

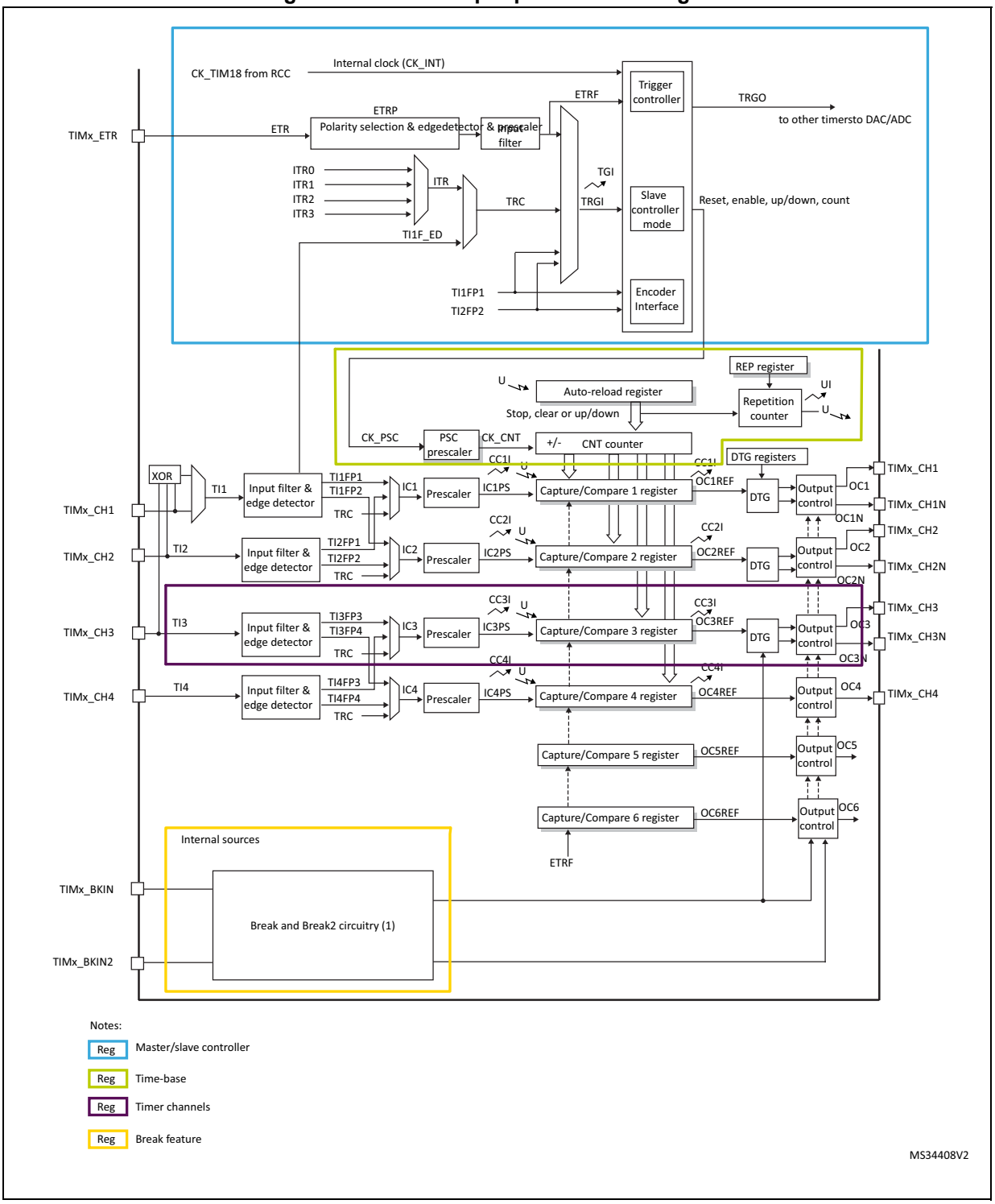
arm

a. Arm is a registered trademark of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The Arm word and logo are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved.

Figure 1 shows the block diagram for the TIM1 timer peripheral. The STM32 timer peripheral is made by the assembly of four units:

1. The master/slave controller unit
2. The time-base unit
3. The timer channels unit
4. The Break feature unit.

Figure 1. TIM1 timer-peripheral block diagram



1.2.1 The master/slave controller unit

The master/slave unit provides the time-base unit with the counting clock signal (for example the CK_PSC signal), as well as the counting direction control signal. This unit mainly provides the control signals for the time-base unit.

The master/slave controller decides the right counting configuration for the time-base unit based on the timer master/slave configuration; it also decides the actual counting status.

For example, if the timer is configured in one of the encoder modes by writing the right value into the SMS control bit-field within the TIMx_SMCR timer register, then the counting clock signal and the counting-direction control signal is computed based on the state of the TI1FP1 and TI2FP2 input signals.

The master/slave controller unit handles the inter-timers synchronization. This unit can be configured to output a synchronization signal (TRGO signal) next to a certain timer internal event. It can be configured as well to control the time-base counter in function of external events (like internal events of other timers or external signals).

It is possible to configure one slave timer to increment its counter based on a master-timer events such as the timer update event. In this example the master-timer event is signaled by the master timer master/slave controller unit. This controlling unit uses the master timer output-TRGO signal. The master timer output-TRGO signal is connected to the slave timer TRGI-input signal. The master/slave controller unit of the slave timer is configured to use the TRGI-input signal as clock source to increment the slave timer counter.

Not all the STM32 timer peripherals feature the same master/slave controller capabilities. The TIM1 timer peripheral taken as example, embeds the full master/slave capabilities; contrary to the TIM6 and TIM7 basic timers that embed the simplest master/slave controller. The master/slave controller within the TIM6 and TIM7 timer peripherals has no control bit-filed.

For the TIM6 and TIM7 timer peripherals, the time-base counter is always up-counting with no means to reset its content next to external events. It is not possible either to clock them with a different clock source nor with the timer peripheral internal clock.

1.2.2 The time-base unit

The time-base unit is made by the timer counter in addition to a prescaler stage and a repetition counter. The clock signal fed into the time-base unit passes first through a prescaling stage before reaching the time-base counter.

Depending on the content of the TIMx_PSC timer prescaler register, the counting signal frequency may be scaled down before reaching the counter stage. The output signal of the prescaling stage is the clock counting signal for the timer counter stage.

The timer counter is controlled by two timer registers:

- The TIMx_CNT timer register is used to read and write the content of the timer counter.
- The TIMx_ARR timer register contains the reload value of the timer counter.
  - If the timer counter is up-counting and it reaches the content of the timer auto-reload register (TIMx_ARR), then the timer counter resets itself and a new counting cycle is restarted.
  - If the timer counter is down-counting and it reaches the zero value, then the timer counter value is set to the content of the timer auto-reload register (TIMx_ARR) and a new counting cycle is restarted.

Each time a new counting cycle is restarted, a timer “update event” is triggered as long as the content of the repetition counter is null. If the content of the repetition counter is not null, then no “update event” is triggered, but a new counting cycle is restarted and the content of the repetition counter is decreased by one. Next to each “update event” the content of the repetition counter is set to the value stored by the TIMx_RCR timer register.

Not all the STM32 timer peripherals embed the repetition counter. If the repetition counter is not embedded, then the timer peripheral would behave as if the repetition counter is embedded but its content is null.

1.2.3 The timer-channels unit

The timer channels are the working elements of the timer; they are the means by which a timer peripheral interacts with its external environment. In general, the timer channels are mapped to the STM32 microcontroller pins with few exceptions such as the timer channel 5 and 6 for the TIM1 timer peripheral on the STM32F30x microcontrollers family. A timer channel mapped to an STM32 microcontroller pin can be used either as an input or as an output.

Timer channel configured as output

When configured as an output, the timer channel is used to generate a set of possible waveforms. As long as the channel is configured in output mode, the content of the TIMx_CCRy channel register is compared to the content of the timer counter.

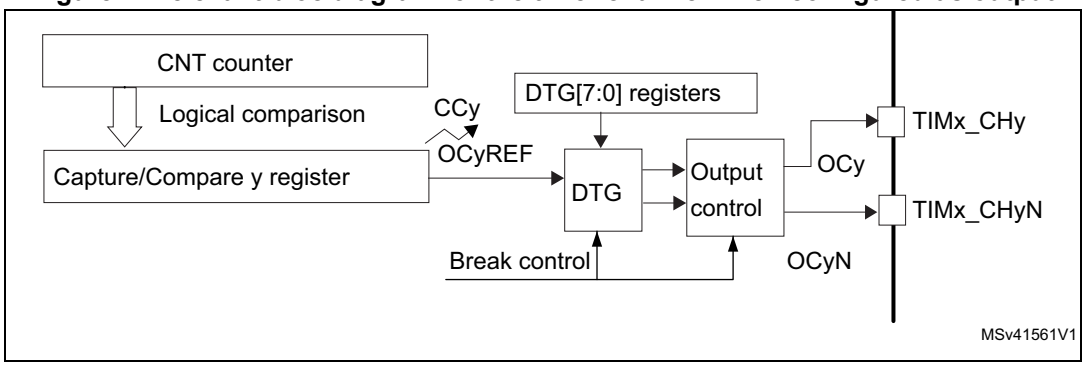
Based on the results of this continuous logic comparison and based on the configured output sub-mode (like PWM1 mode or Inactive mode), the timer channel internal output OCyREF is either set or reset.

The timer channel internal output OCyREF is then fed into the channel output stage. The channel output stage applies a set of conditioning operations on the OCyREF signal based on a set of configured parameters (like channel polarity configuration or dead-time generation among others).

The output signal of the channel output stage is mapped to the microcontroller pins as alternate function. Note that some output stages of timer channels, as the one represented in *Figure 2*, may output two complementary signals.

The control bit-fields for such channel output stage provides the means to configure each output signal separately (like enabled/disabled output signal or like polarity).

Figure 2. Relevant bloc diagram for the timer channel when configured as output



Timer channel configured as an input

When configured as an input, the timer channel can be used to time-stamp the rising and/or the falling edge of external signals. To handle this function, the channel input is mapped to one of the microcontroller pins.

Some timer-channel inputs are mapped as well to some on-chip signals, for example the oscillator output for calibration purposes. The TIy timer channel input feeds the channel input conditioning circuitry as shown in *Figure 3*. The conditioning circuitry includes a filtering stage and an edge detector. The filter stage rejects pulses with duration less than the configured one. The edge detector detects if an active edge occurred on the concerned timer input after filtering.

The active edge configuration is set by writing to the channel polarity control bit-fields within the TIMx_CCER timer register. The conditioning circuitry outputs two signals:

- The TIyFPy: is the TIy timer input signal which was filtered and on which an active edge is detected depending on the polarity of the timer channel “y”.
- The TIyFPz: is always the TIy timer input signal which was filtered but on which an active edge is detected depending on the polarity of the timer channel “z”.

The TIyFPz signal is redirected to the prescaler input of the channel “z” where the TIzFPy signal is redirected to the prescaler input of the channel “y” as shown in *Figure 3*. This cross swap of filtered input signals is very useful for time-stamping both the rising and the falling edges of an input signal. It is very useful for implementing PWM (pulse width modulation) input applications.

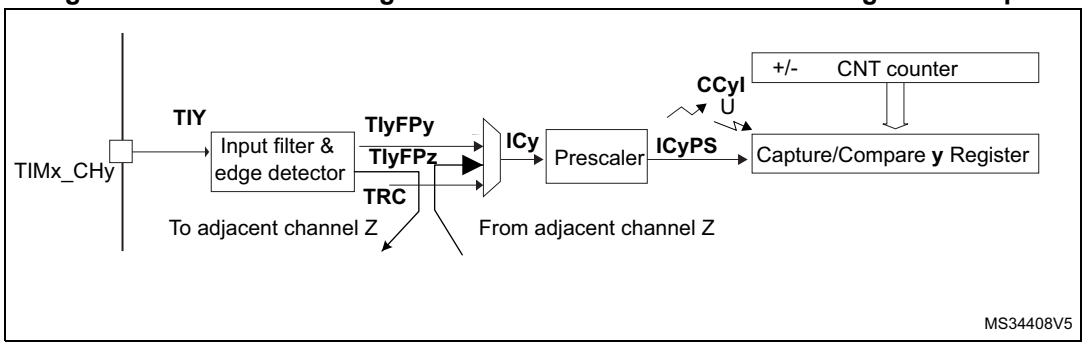
Each timer channel can be configured in one of three possible input modes. Each input mode corresponds to one of the possible three inputs of the prescaler mux connected to the timer channel prescaler. The CCyS control bit-field controls if the timer channel is configured in output mode (like CCyS[1:0] = ‘00’) or if it is configured in one of the input modes (like CCyS[1:0] different from ‘00’).

The same TIMx_CCMRn timer registers (‘n’ can be any number, but generally is either 1 or 2) are used to configure the timer channels either as input or as output. Some control bit-fields of the TIMx_CCMRn timer registers have different interpretation based on the channel configuration, input or output modes.

The timer channel prescaler may be configured to scale down the frequency of the active edges detected on the timer input TIy. The detection of an active edge on the output of the channel prescaler triggers the transfer of the timer counter content to the “y” register of the TIMx_CCRy timer channel.

The content of the “y” register of the TIMx_CCRy timer channel is the timestamp of the last detected active edge on the output of the channel “y” prescaler. It is the timestamp of the last detected active edge on the TIy timer input if the channel “y” prescaler is configured to not scale down the input signal (for example when the prescaler ratio = 1, in other words, channel prescaler is bypassed).

Figure 3. Relevant bloc diagram for the timer channel when configured as input



1.2.4 The Break feature unit

The Break feature unit is embedded only by timer peripherals that feature complementary outputs. In other words, only timer peripherals that have at least one channel with two complimentary outputs embed the Break feature.

The Break feature acts on the output stage of timer channels configured in output mode. As soon as an active edge is detected on the break input, the outputs of timer channels configured in output mode are either turned off or forced to a predefined safe state.

The Break feature is typically used for implementing safe shutdown functionality in electrical power inverters next to anomalies.

The application note *Using STM32 device PWM shut-down features for motor control and digital power conversion* (AN4277) provides a detailed description of the Break feature across the STM32 microcontroller families.

1.3 STM32 timer peripheral basic operating modes

This section provides a set of source code snippets for a set of basic configurations of the STM32 timer peripheral. These snippets were developed using the C programming language.

1.3.1 Timer time-base configuration

```
#define ANY_DELAY_RQUIRED       0x0FFF
/* Hardware-precision delay loop implementation using TIM6 timer
peripheral. Any other STM32 timer can be used to fulfill this function, but
TIM6 timer was chosen as it has the less integration level. Other timer
peripherals may be reserved for more complicated tasks */
/* Clear the update event flag */
TIM6->SR = 0
/* Set the required delay */
/* The timer presclaer reset value is 0. If a longer delay is required the
presacler register may be configured to */
/*TIM6->PSC = 0 */
TIM6->ARR = ANY_DELAY_RQUIRED
/* Start the timer counter */
TIM6->CR1 |= TIM_CR1_CEN
```

```
/* Loop until the update event flag is set */
while (!(TIM6->SR & TIM_SR_UIF));
/* The required time delay has been elapsed */
/* User code can be executed */
```

1.3.2 Timer channel-configuration in input mode

```
/* Variable to store timestamp for last detected active edge */
uint32_t TimeStamp;
/* The ARR register reset value is 0x0000FFFF for TIM3 timer. So it should
be ok for this snippet */
/* If you want to change it uncomment the below line */
/* TIM3->ARR = ANY_VALUE_YOU_WANT */
/* Set the TIM3 timer channel 1 as input */
/* CC1S bits are writable only when the channel1 is off */
/* After reset, all the timer channels are turned off */
TIM3->CCMR1 |= TIM_CCMR1_CC1S_0;
/* Enable the TIM3 channel1 and keep the default configuration (state after
reset) for channel polarity */
TIM3->CCER |= TIM_CCER_CC1E;
/* Start the timer counter */
TIM3->CR1 |= TIM_CR1_CEN;
/* Clear the Capture event flag for channel 1 */
TIM3->SR = ~TIM_SR_CC1IF;
/* Loop until the capture event flag is set */
while (!(TIM3->SR & TIM_SR_CC1IF));
/* An active edge was detected, so store the timestamp */
TimeStamp = TIM3->CCR1;
```

1.3.3 Timer channel-configuration in output mode

```
/* The ARR register reset value is 0x0000FFFF for TIM3 timer. So it should
be ok for this snippet. If you want to change it uncomment the below line */
/* TIM3->ARR = ANY_VALUE_YOU_WANT */
/* The TIM3 timer channel 1 after reset is configured as output */
/* TIM3->CC1S reset value is 0 */
/* To select PWM2 output mode set the OC1M control bit-field to '111' */
TIM3->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2;
/* Set the duty cycle to 50% */
TIM3->CCR1 = TIM3->ARR / 2;
/* By default, after reset, preload for channel 1 is turned off */
/* To change it uncomment the below line */
/* TIM3->CCMR1 |= TIM_CCMR1_OC1PE;
/* Enable the TIM3 channel1 and keep the default configuration (state after
reset) for channel polarity */
TIM3->CCER |= TIM_CCER_CC1E;
/* Start the timer counter */
```

```
TIM3->CR1 |= TIM_CR1_CEN
```

1.4 STM32 timer peripheral advanced features

This section provides a detailed description of some common STM32 timer features used by application examples detailed later in this document.

1.4.1 Filtering stage

Timer inputs (like ETR input or channel inputs) feature a filtering stage that may be activated to filter out external signal pulses with duration less than a desired threshold.

The maximal duration of filtered pulses depends on two parameters:

- The configuration of the filtering stage respective to a certain timer input. For example, the ETR input filtering stage is configured through the ETF[3:0] control bit-field within the TIMx_SMCR register. The configuration of an input filtering stage implies the selection of a sampling clock source, the setting of the sampling clock frequency and the setting of the minimal time duration of a valid pulse in units of clock cycles of the already configured sampling clock.
- In the case that the FDTS clock source is used as sampling clock source, the filtering stage minimal valid pulse duration is impacted by the value written to the CKD[1:0] (or/and CKD2[3:0][(a)]) control bit-field within the TIMx_CR1 register. The FDTS (or/and FDTS2[(a)]) clock signal is derived from the timer clock signal, and the CKD[1:0] (or/and CKD2[3:0][(a)]) control bit-field sets the ratio between these two clock signals.

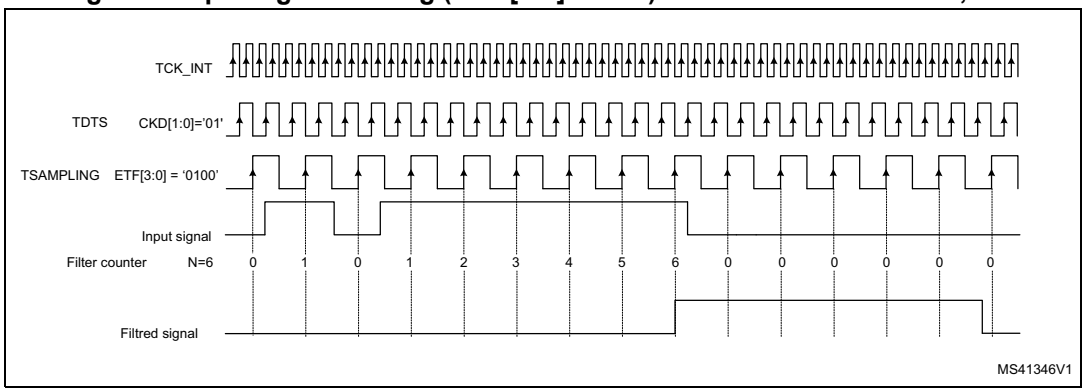
One out of two clock sources can be used as sampling clock source, either the timer clock signal FCK_INT or the FDTS clock source

Figure 4 shows a practical example where the filtering stage is activated for the timer input ETR. For the sake of this demonstrative example:

- The timer clock frequency is FCK_INT = 1MHz
- The CKD [1:0] = ‘01’. It means that the FDTS clock signal frequency is two times less than the timer clock signal frequency: Fdts = Fck_int/2 = 500 kHz.
- The ETF [3:0] = ‘0100’. It means that the FDTS clock signal is selected as the sampling clock for the filter with its frequency scaled down by factor of two. It also means that a valid pulse on the ETR input shall be at least 6 sampling clock cycles long.
- For this example, any pulse signal on the timer ETR input with a duration shorter than 6x Tsampling = 6x 1/250 kHz = 24 µs is rejected.

a. Only available in the STM32C5 and STM32G5 series.

Figure 4. Input signal filtering (ETF [3:0]= 0100) : FSAMPLING = FDTS/2, N=6



1.4.2 The preload feature of the timer registers

The preload feature in the context of the STM32 timer peripheral, refers to the duplication of some timer registers or some timer control bit-fields. As the content of some timer registers and some control bit-fields impacts directly the waveforms outputted by the timer channels, the update of the content of those registers and control bit-fields should be perfectly synchronized with the timer “update event” triggered next to a new counting-cycle start. This tight synchronization would be practically impossible to achieve, if those registers and control bit-fields were not Preload.

When a timer register features the preload capability, two physical instances of that timer register exist:

- The active register instance (also called the shadow register instance): its content is used by the timer peripheral logic to generate the timer channel outputted waveforms.
- The preload register instance: this is the register instance accessed by the software when the preload feature of the concerned register is enabled.

If the preload feature is turned off as shown in *Figure 5*, there are two main characteristics to highlight:

- The preload register instance looks as inexistent
- Any write access to the concerned register by software is performed on the active register instance.

If the preload feature is enabled as shown in *Figure 6*, the two characteristics to highlight are:

- Any access to the concerned register is performed on the preload register instance while the content of the active register instance is not changed.
- As soon as an “update event” is generated within the timer peripheral, the content of the preload register instance is transferred to the active register instance immediately. The content of the preload register is transferred in perfect synchronization with the “update event”, in other words, it is in perfect synchrony with the new counting cycle corresponding to a new cycle in the outputted waveform.

Figure 5. Preload mechanism for timer channel register - disabled

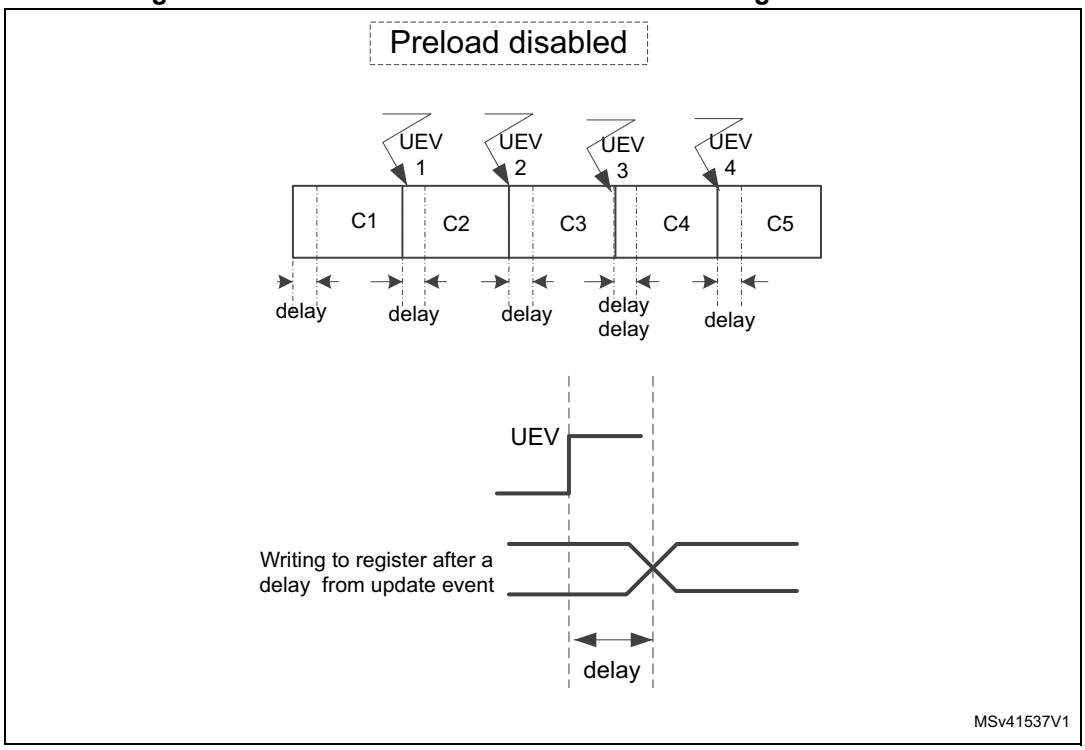
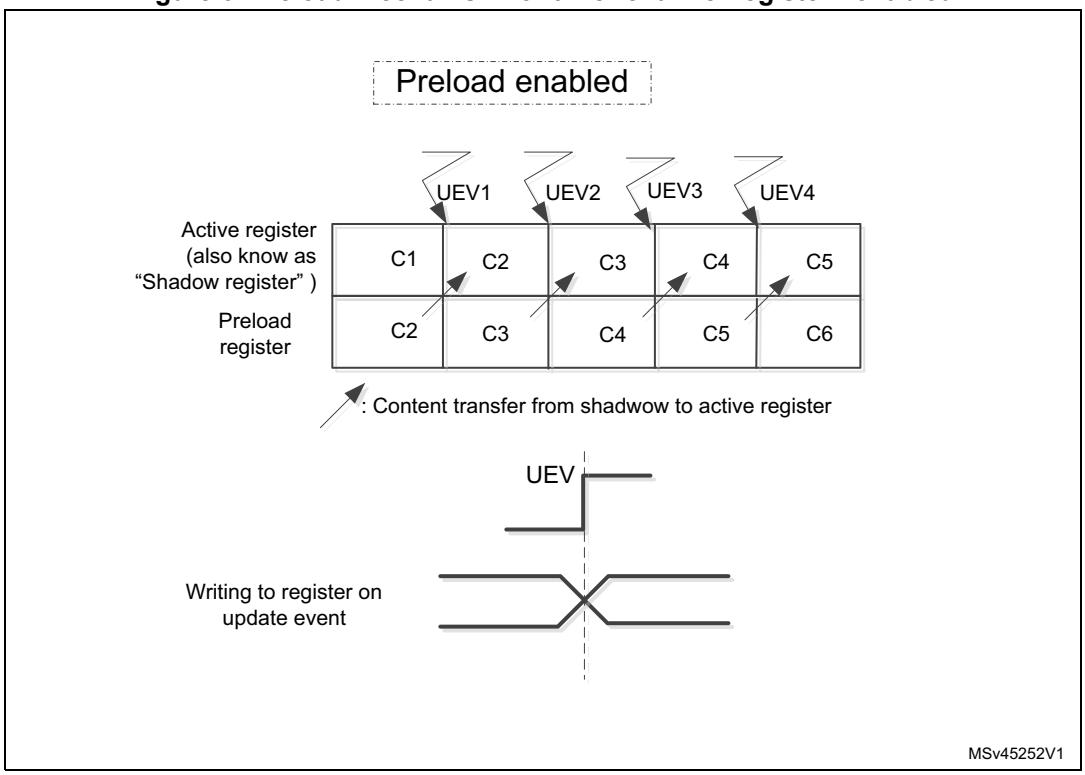


Figure 6. Preload mechanism for timer channel register - enabled



The preload feature is available for:

- The auto-reload timer register (TIMx_ARR)
- The timer prescaler register (TIMx_PSC) (cannot be turned off)
- The timer channel registers (TIMx_CCRy)
- The CCxE and CCxNE control bit-fields within the TIMx_CCER timer register
- The OCxM control bit-field within the TIMx_CCMRn timer registers.

The preload feature can be of big interest when outputting a PWM signal on a certain timer channel. The channel output level depends on the continuous comparison between the timer counter value and the TIMx_CCRy timer channel register value, this is why any change to the timer channel register may induce an immediate timer channel output level change.

As a consequence, writing directly to the channel register in the middle of a PWM period may generate spurious waveforms. To overcome this problem, the preload feature should be enabled for the concerned timer channel register. When the preload feature is enabled, any write access is made to the channel preload register instance while the channel active register instance remains unchanged. There is no perturbation on the ongoing PWM period.

As soon as an “update event” is generated within the timer, the preload register instance content is transferred into the active register instance. The active register instance is used during the next PWM period to carry the comparison operation and to define the new duty-cycle ratio.

An update overrun status bit[a] (UIOVRF in the TIMx_SR register) indicates that an update request occurred while an update interrupt was still pending (update interrupt flag set).

This bit is useful when debugging closed-loop applications, as it indicates that:

- the update interrupt service routine is too long with respect to the interval between consecutive update events, or
- the update interrupt service routine was started too late with respect to the subsequent update event.

In summary, the preload feature guarantees that no spurious waveforms are generated while accessing the timer registers and control bit-fields that have direct impact on the timer channels output; especially in the middle of a waveform output cycle.

a. Only available in the STM32C5 and STM32G5 series.

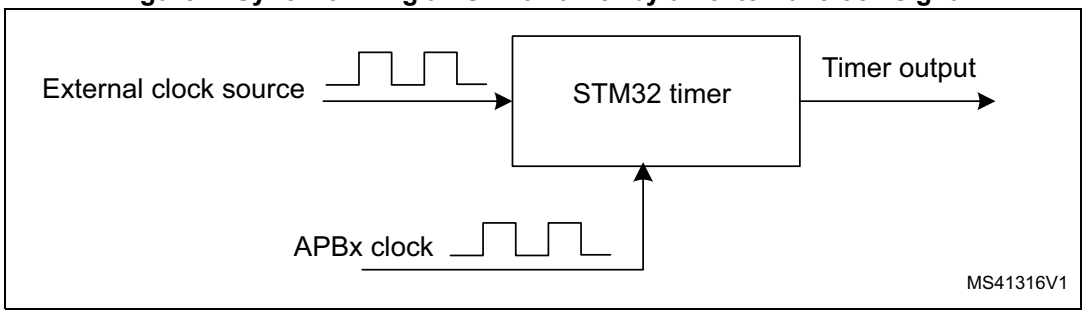
2 Timer clocking using external clock-source

2.1 Overview

The STM32 timer peripherals can be clocked by an external source clock, but it does not mean that the APB (advanced peripheral bus) clock is not required. An STM32 timer peripheral synchronizes the external clock signal with its own core clock (which is the APB clock). The resulting synchronized clock signal is fed into the timer's prescaler which by turn feeds the timer's counter.

An STM32 timer peripheral requires two clock sources to keep updating its time base continuously (see *Figure 7*). The external clock-source period is the time unit used to update the timer peripherals' time base.

Figure 7. Synchronizing an STM32 timer by an external clock-signal

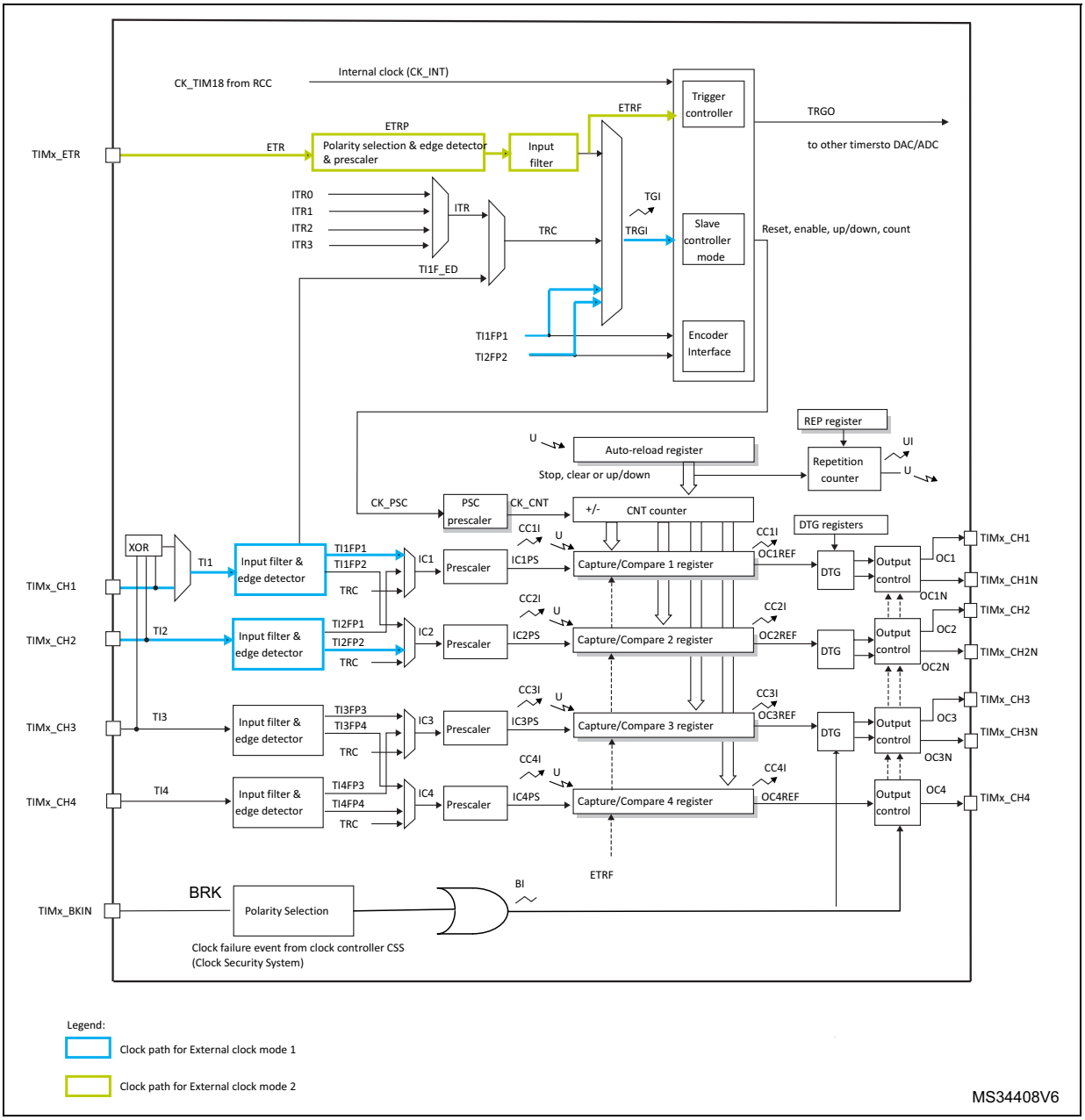


There are two ways to synchronize (or externally clock) an STM32 timer:

- External source clock mode 1: by feeding the external clock signal into one of the timer channel inputs, TIx inputs.
- External source clock mode 2: by feeding the external clock signal into the timer ETR input (if it is implemented and available).

The *Figure 8* below shows the clock path for both external clock-source modes.

Figure 8. Clock path for external clock-source modes



2.2 Synchronization block

Before introducing the modes of timer peripheral clocking by an external clock-source, it is important to first introduce the synchronization mechanism implemented by the STM32 timer peripherals when dealing with external signals. External signals are the signals coming from outside of the timer peripheral. They might be synchronized with a clock signal different from the one used by the timer peripheral or they might be fully asynchronous.

A timer peripheral needs to deal with and handle external signals that may be fully asynchronous and adjust its outputs state or waveform accordingly. A timer peripheral also

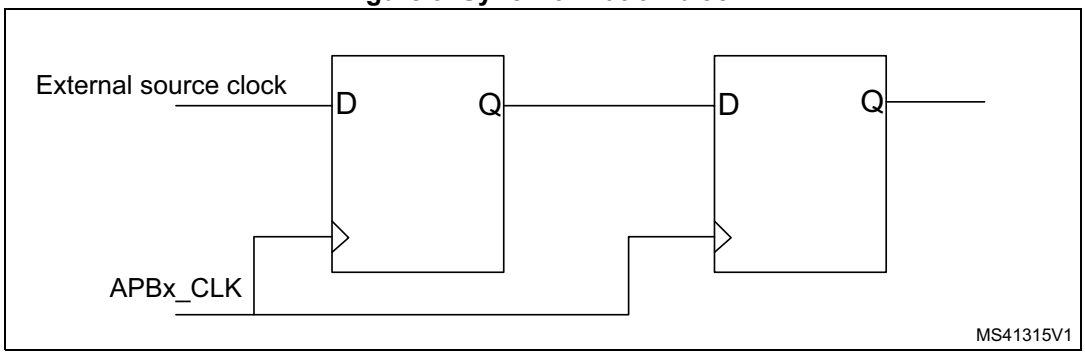
needs to be able to inform the software about the timestamp on which a signal on a given timer input changed its state (for example a signal toggling).

To deal with a fully asynchronous signal, a timer peripheral needs first to resynchronize it with its own clock signal. It might need to resynchronize for example with the timer core logic clock signal before feeding the resulting signal to the different sub-blocs of the timer peripheral. This action protects the timer core logic from going into metastability issues.

Figure 9 shows the synoptic diagram for the synchronization circuit used to synchronize external signals fed into the timer inputs. The synchronization circuit is mainly composed of two cascaded D flip-flops that are clocked by the timer core clock signal. The external signal is fed into the first stage D flip-flop input where the synchronized signal is retrieved from the output of the second D flip-flop. This synchronization circuit introduces a delay of at least two timer core clock cycles and at most three clock cycles.

The timer embeds “synch first” on all inputs except on the ETR where there is a prescaler first then re-synch.

Figure 9. Synchronization block



As shown in the formula below, the frequency of the input signal must be smaller than three times the frequency of timer core clock signal.

$$Freq_{TIMCLK} \geq 3 \times Freq_{inputsignal}$$

2.3 The external clock-source mode 1

When the external clock-source mode 1 is activated, any signal that can be routed into the TRGI internal timer signal can as well clock the timer counter. *Figure 8* shows the possible clock source for the timer counter, which are:

- The ETRF input signal: ETR signal after being prescaled, synchronized then filtered.
- The inter-timer peripherals synchronization signals (ITR inputs)
- The TI1FD signal which is the output of timer channel1 but which sensitive to both signal edges (each transition of the timer input 1 generates a pulse)
- The TI1FP1 and TI2FP2 input signals that are the synchronized, filtered then the TI1 and TI2 prescaled timer inputs, respectively.

This section deals only with the external clock mode 1 when an external source clock is fed into the timer through one of its inputs, in other words when a clock is fed into the timer through the ETR, TI1 or TI2 timer inputs.

Using the ETR input for feeding the timer is an alternative configuration for the external clock-source mode 2, so it is not detailed further within this section. The corresponding reference manual contains the right configuration to activate this clocking alternative.

Timer inputs, TI1 and TI2 as clock source

Only TI1 and TI2 inputs can be used to feed a clock signal into the timer counter when the external clock mode 1 is activated. If the timer does already embed four channels, the TI3 and TI4 inputs cannot clock the timer in this mode.

The clock signal fed into the TI1 or TI2 timer peripheral inputs is first conditioned before reaching the timer counter. The input conditioning is made of an external signal synchronization stage then a prescaling and filtering stage.

The input filter and prescaler are configurable. The channel associated with the targeted timer input should be configured as an input channel; then the same control-bit fields used for configuring an input channel are used to configure the input for the external clock-source.

Below is the typical sequence to configure the TI1 or TI2 inputs as input for an external clock signal:

1. Configure the channel associated with the timer input that feeds the external clock signal as an input channel. Even though any value written to the capture/compare selection (CCxS) control bit-field except the 00 value sets the timer channel in input mode, the right value to configure is CCxS = 01.

Note: The CCxS bit-field is writable only when its associated timer channel is disabled (for example when. the enable/disable control bit-field for that channel in the TIMx_CCER register is reset, CCxE = '0'.

2. Configure the active edge polarity. Some of the STM32 microcontrollers embed timers with inputs sensitive for either falling, rising or both edges (like STM32F2 microcontrollers); other STM32 families embed timers that have their inputs only sensitive for either rising or falling edges (like STM32F1 microcontrollers). The reference manual document for each STM32 microcontroller specifies the right value to write into the polarity control bit-fields to configure the convenient timer input polarity. As an example, to make the timer input on STM32F30x microcontrollers sensitive to both edges, the polarity control bit-fields for the concerned channel should be configured as CCxP = 1 and CCxNP = 1.
3. If needed, configure the input filter associated with the concerned channel to reject pulses with duration shorter than a given value. The threshold for rejecting or passing the input pulse is configured through the ICxF[3:0] bit-field within the TIMx_CCMRx register associated to the used timer channel.
4. After having well configured the timer input with the required setting, redirect the conditioned input signal into the timer counter. This step is done by writing the right value into the trigger selection control bit-field TS[2:0] within the TIMx_SMCR register. For instance, if the TI1 timer input is used as the external clock signal input, then the right configuration should be TS[2:0]=101.
5. Finally, activate the external clock-source mode 1. This is done by writing the right value into the slave/master selection (SMS) bit-field within the TIMx_SMCR register. Note that this bit-field has different widths along the STM32 microcontroller families. The reference manual document specifies the right value to write to the SMS bit field in

order to activate the different clock modes. For example, on the STM32F1 series, the SMS bit filed is 3-bit wide and the right value to configure is SMS[2:0] = 111, whether for the STM32F30x microcontrollers the SMS bit-field is 4-bit wide and the right value to configure is SMS[3:0] = 0111.

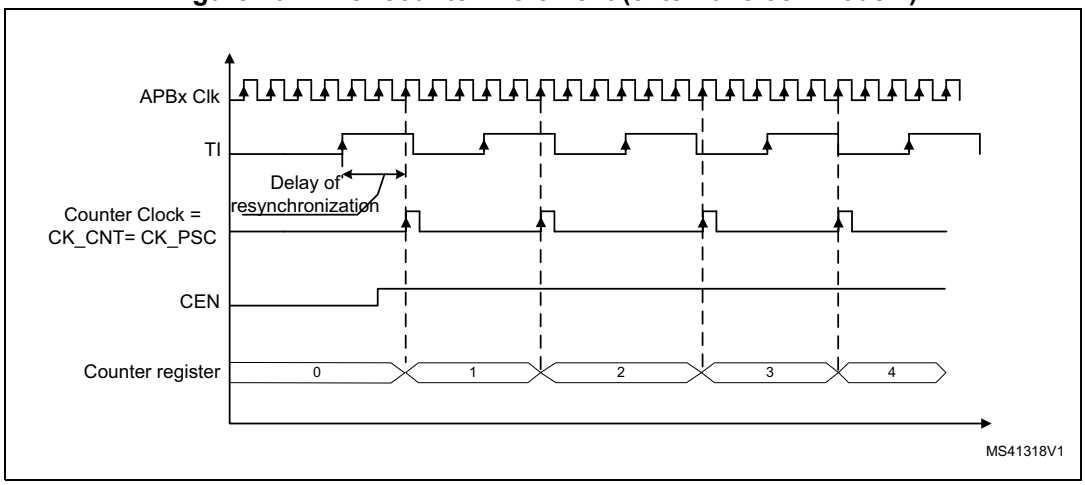
6. Set the CEN control bit-field within the TIMx_CR1 register to enable the timer counter.

Figure 10 shows the typical counting sequence of the timer counter where the timer peripheral is configured in up-counting configuration. The counter content is null at the time of setting the CEN control bit-field, and the timer peripheral is externally clocked in mode 1 through its TI1 input.

This example shows that the configured active edge is the rising edge where the filter and prescaler control bit-fields are kept at their default value (like status after reset). *Figure 10* also shows the effect of the resynchronization stage at the TI1 timer input.

There is a delay between the rising edge of the clock signal on the TI1 timer input and the rising edge of the internal counting signal fed into the timer counter CK_CNT. In this example the TIMx_PSC timer prescaler register content is null, so the CK_CNT signal is the same as CK_PSC signal (CK_CNT = CK_PSC).

Figure 10. Timer counter increment (external clock mode 1)



Due to the resynchronization stage, it is required that the external clock-source frequency is less than twice the timer frequency, as shown at the formula below:

$$TIMx_{CLK}freq \geq 3 \times ITxfreq$$

2.4 The external clock-source mode 2

When the external clock-source mode is activated, the timer counter is updated on the active edges detection of the clock signal fed through the ETR timer input. For some of the timer peripherals on certain STM32 families, the ETR input may not be routed out of the microcontroller.

For some ETR, the input is not mapped to a microcontroller IO as an alternate function. In some other timer peripherals the ETR timer input is multiplexed with the channel1 timer input.

The main advantage of using the external clock-source mode 2 compared to mode 1 is that the externally provided clock signal frequency can be equal or even higher than the frequency of the internal timer core clock (like on APB bus clock).

This does not mean that the timer counter can be updated with a cadence higher than the timer core clock's cadence (for example, incremented cadence if the timer is configured in up-counting mode).

The ETR timer input is the only timer input that features a presacler stage before the resynchronization stage. This presacaler stage is fully asynchronous and can be divided down to eight times less the asynchronous input signal frequency.

In some other STM32 microcontrollers, such as STM32C5 and STM32G5, in addition to the asynchronous prescaler (division by 1, 2, 4, or 8), which can process tim_etr_in signals with frequencies higher than tim_ker_ck, there is also a linear synchronous prescaler (division by 1, 2, 3, …, 15).

This synchronous prescaler can only operate with tim_etrp signal frequencies lower than tim_ker_ck/3.

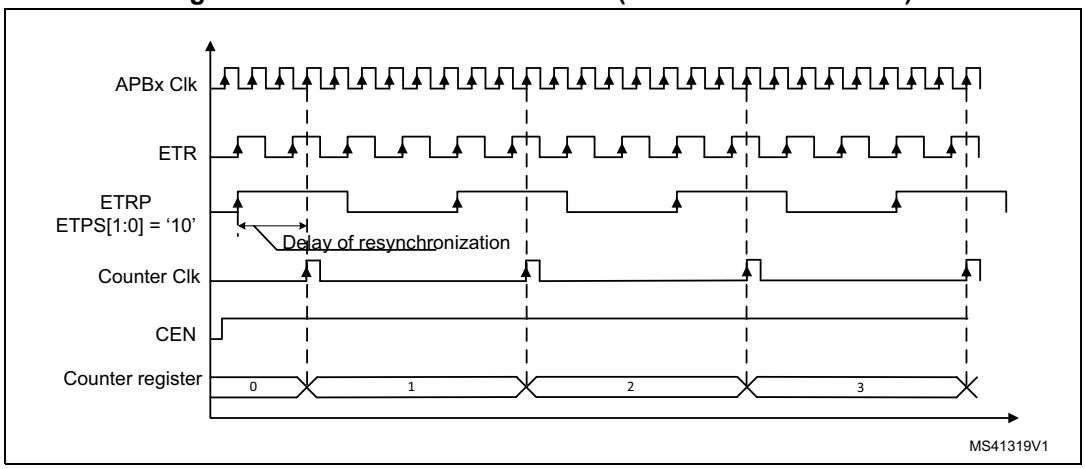
The output of the asynchronous prescaler stage, named ETRP signal, is routed to the resynchronization stage (or to the synchronous prescaler then to the resynchronization stage) before being fed into the timer counter. The output of the resynchronization circuit is called ETRF signal. The ETRP signal has the same constraint as any other asynchronous signal fed to the timer, its frequency should be less than three times the timer internal core clock frequency.

This feature of the external clock-source mode 2 is of great interest in many applicative cases. For instance, it may be required to count the number of pulses generated by a certain sensor, but that sensor output frequency is higher than the timer internal core clock. In that case, the prescaler should be configured to scale down the input signal frequency by a given ratio that makes it compatible with the resynchronization stage constraint on external signal input frequency. When interpreting the counted pulse number, it should be taken into account that the signal frequency was divided by a given ratio.

Figure 11 shows an example where the timer peripheral is configured in external clock-source mode 2 and where the clock signal is fed into the timer through its ETR input. In this example the external clock signal frequency is higher than the timer internal core clock (the APB bus clock). The ETR asynchronous prescaler was configured to divide the input signal frequency by four by setting the external trigger prescaler control bit-field within the TIMx_SMCR register to ETPS = '10'.

Also, *Figure 11* shows a delay between the output of the prescaler (ETRP signal) and the counter clock signal CK_CNT used to update the timer counter. This delay is inserted by the resynchronization stage.

Figure 11. Timer counter increment (external clock mode 2)



Below is the typical and recommended sequence to configure the timer peripheral in external clock-source mode 2:

1. Evaluate the maximum frequency of the input clock signal and choose the asynchronous prescaler ratio. If the external clock signal frequency is less than the timer internal core clock by more than three times, then the use of the prescaler is optional and may not be used. In the case that the asynchronous prescaler is not used, the ETPS control bit-filed should be reset (the ETPS bit-filed is by default reset). In the case that the synchronous prescaler is not used, the SETPS control bit-filed should be reset (the SETPS bit-filed is by default reset)
2. If required, activate the filtering stage to reject clock signal pulses with duration shorter than a certain threshold. For more information on how to set up the filter feature on the STM32 timer peripheral external inputs, refer to *Section 1.4.1: Filtering stage on page 14*.
3. Configure the active edge of the external clock signal. The ETP control-bit sets for which the external clock-source mode 2 edge is sensitive. By default and after reset, the ETP control-bit is reset, which makes the external clock signal rising edges for the active ones (for example rising edges of the external clock signal triggers the timer counter update).
4. Enable the external clock-source mode 2 by setting the external clock enable control bit, ECE =1.
5. It is mandatory, at the end to set the counter enable control bit within the TIMx_CR1 register to enable the timer counter.

Note: The ETR input can be used also as an input for the external clock signal when the external clock-source mode 1 is configured. It is possible to activate both the external clock-source mode 1 and 2 simultaneously and make them use simultaneously the ETR input. In that case the priority is given to the external clock-source mode 2 and it is the one used for clocking the timer counter.

2.5 External clock-source mode 1 versus mode 2

Both external clock-source modes, 1 and 2, seem to have the same functionality but studying them carefully shows that they differ in some specific characteristics that makes each one of them suitable for a different range of applicative cases.

See below the main differences between mode 1 and 2:

- It is possible to use the external source clock mode 1 to update the timer counter on both edges of the external clock signal. This is not possible when the external clock-source mode 2 is used.
- By using external clock-source mode 2, it is possible to clock the timer peripheral by an external clock-source. It is then still possible to configure the timer peripheral, at the same time, in one of the compatible timer slave modes. For example, if it is required to count the number of pulses generated by certain sensor within a given period of time repeatedly:
  - Make one timer peripheral, like TIMy, count the number of pulses generated by the sensor by configuring it in external clock-source mode 2.
  - By configuring a second timer peripheral, like TIMz, to generate a trigger-out signal (TRGO) each given period of time, repeatedly.
  - The first timer (TIMy) should be configured also in slave reset mode and use the TRGO output signal of the second timer (TIMz) as trigger for the reset.

One way to make a timer peripheral output a pulse on its TRGO output signal at regular time intervals is by setting the TIMx_ARR timer register to a certain value so a periodic timer “update event” is generated.

A timer “update event” can trigger a pulse on the timer’s TRGO signal if the timer master-mode is set to “update value” (for example if MMS[2:0] bit-filed is set to ‘010’ within the TIMx_CR2 register).

A timer may be configured in reset slave mode by setting the right value in the slave mode selection control bit-field (for example SMS[2:0] = 100 within the TIMx_SMCR register).

Selecting the right trigger for the reset slave mode is ensured by configuring the right value into the trigger selection control bit-field TS[2:0].

The value to write into TS[2:0] control bit-filed depends on which ITR timer input of the first timer (TIMy) the TRGO output of the second timer (TIMz) is internally connected to. The reference manual of each STM32 microcontroller lists all the internal interconnections between timer peripherals.

2.6 Application: timer clocking using external clock-source on ETR timer input

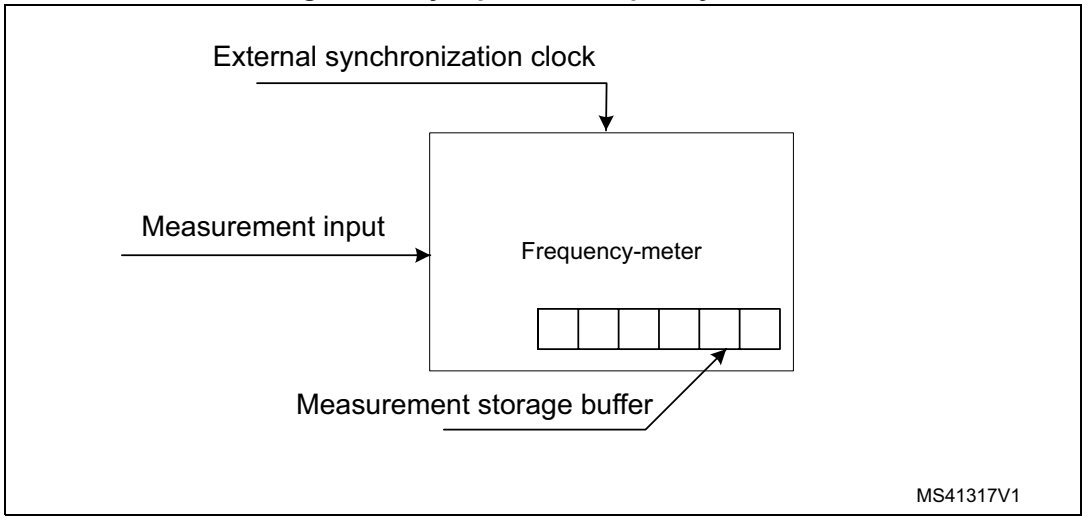
This application describes one use-case of using an external clock-source to clock the timer peripheral. This application is developed around the external clock-source mode 2 but it can be easily reshaped to use the external clock mode 1.

This application is a basic implementation for a frequency counter like reduced frequency range and precision counter. This application was developed under the assumptions below:

- The microcontroller is using its internal high-speed oscillator (HSI oscillator), as a clock source.
- This application is used in an environment where a 10 MHz accurate clock source is available (for example within a measurement bench where several instruments provide a calibration 10 MHz output).
- This application demonstration is split into two parts. The first part is where the frequency meter application is only clocked by the internal HSI oscillator. The

second part is where the frequency counter application has access to a relatively more accurate 10 MHz clock source which is fed to the timer ETR input.

Figure 12. Synoptic of a frequency meters



To emphasize the effect of synchronizing a timer peripheral with an external clock on the measurement accuracy, a reference clock signal is injected on the application measurement input as shown in *Figure 12*.

Then a series of measurements of the frequency of the input signals should be carried in both cases: whether there is an external clock signal provided and whether it is not. The measurements spread provides a good indicator if the usage of the external clock-source has enhanced the performance of this basic frequency meter application or not.

To run this application the following hardware set-up is needed:

- An STM32F302 Nucleo board (NUCLEO-F302R8)
- An USB cable to connect the Nucleo board to the development PC (the USB cable also provides the power supply for the board).
- One or several waveform generators.

To reshape this application source code and debug different configurations a set of software tools are needed:

- A development tool-chain supporting the ST-LINK debugger
- The latest version of STM32CubeMX software tool (v4.10.1 was the latest release when this application was developed).

The TIM2 timer has been chosen because it has the highest counter resolution (32 bits) to take more possible samples of PPM.

The channel 2 has been chosen as an input capture because the ETR pin is mapped in the channel 1 of the TIM2 timer.

The idea of this application is to calculate the period of the incoming signal that is the difference between two successive values of the CCR2 register multiplied by the increment period TIM2 counter, then deduct the frequency and the PPM.

$$ppm = \frac{df \times 10^6}{f} \qquad with \qquad df = fm - f$$

– fm: measured frequency
– f: nominal frequency

Configuration

System clock initialization

Clock initialization is done in the main routine using the SystemClock_Config() function right after the HAL library initialization (HAL_Init).

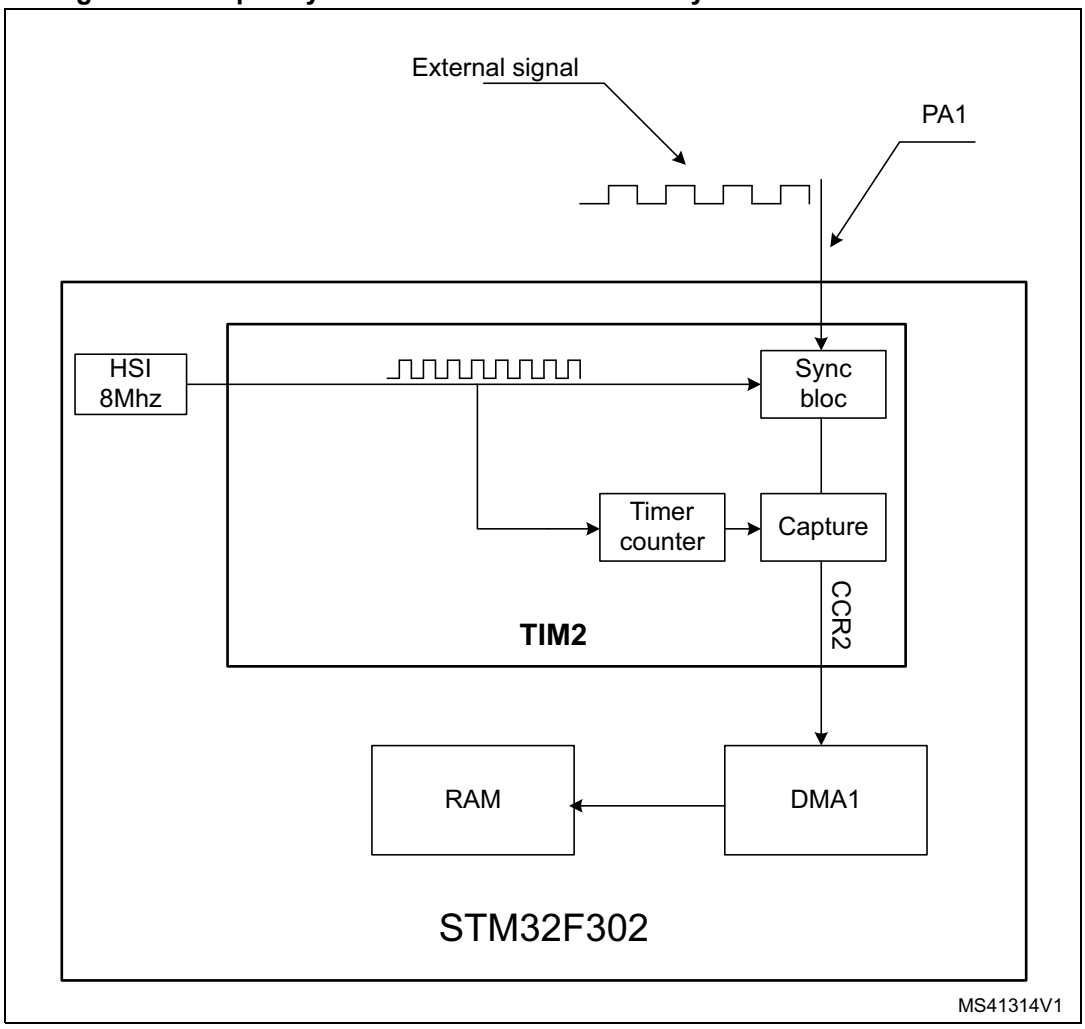
DMA configuration

The DMA is used to copy the values of CCR2 in a buffer to calculate the frequency of the incoming signal.

The DMA is configured as below:

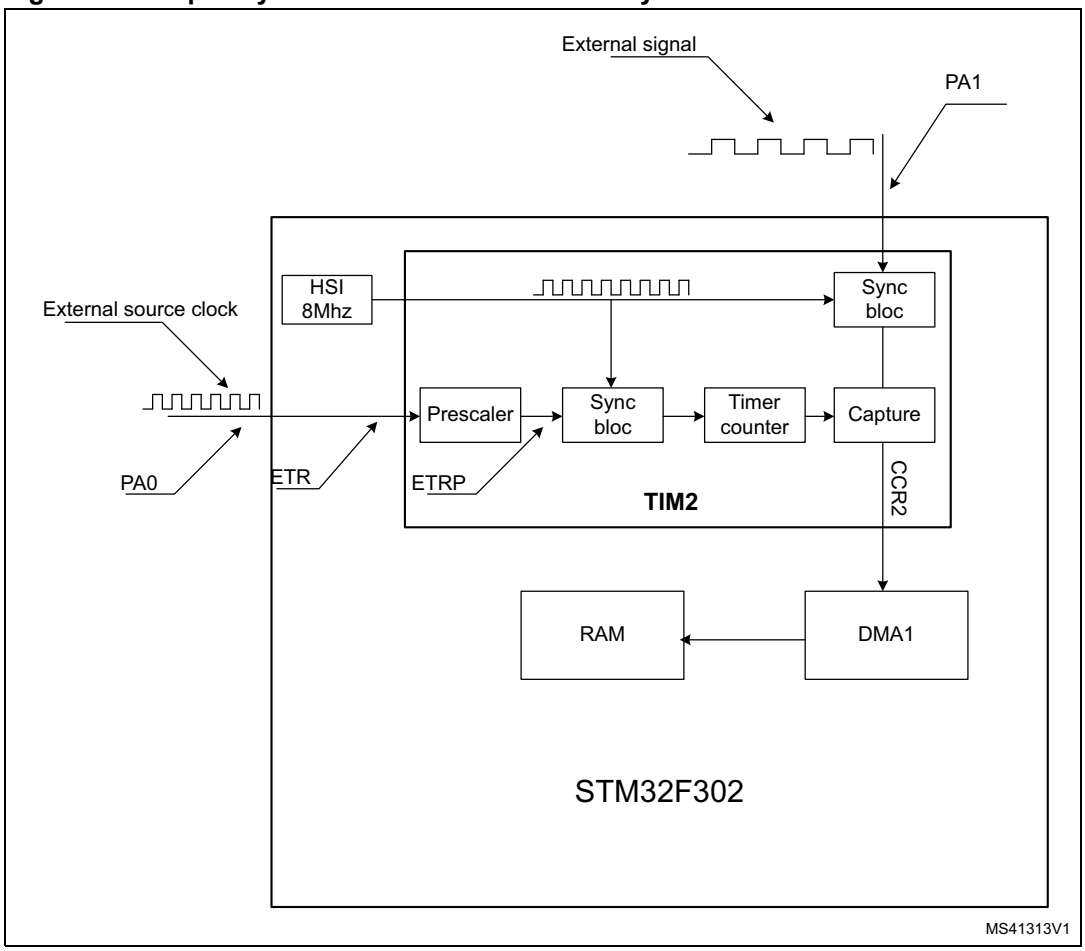
- DMA_Stream = DMA1_Stream6
- DMA_Channel = DMA_Channel_3
- Direction = peripheral to memory
- Incrementation memory= enable
- Peripheral data alignment = alignment word
- Memory data alignment = alignment word
- Mode = normal
- FIFO mode = enable.

Figure 13. Frequency meter architecture clocked by the internal HSI oscillator



In this example, the timer is clocked only by the internal HSI oscillator which frequency is 8 MHz. The measured signal passes first through a timer with the clock synchronization block. Then at each rising edge timer capture, the counter value (available at the CCR2 register) then saves this value in the RAM through the DMA. Finally it applies the formula for deriving the frequency of the input signal.

Figure 14. Frequency meter architecture clocked by the external clock-source mode 2



In this example, the timer is clocked by a calibrated external clock-source from a waveform generator whose frequency is 10 MHz. Then, it passes through an asynchronous prescaler to validate the synchronization condition with the clock timer.

The measured signal passes first through a timer with the clock synchronization block and then at each rising edge timer, capture the counter value which is at the CCR2 register. Then saves this value in the RAM through the DMA. Finally it applies the formula for deriving the frequency of the input signal.

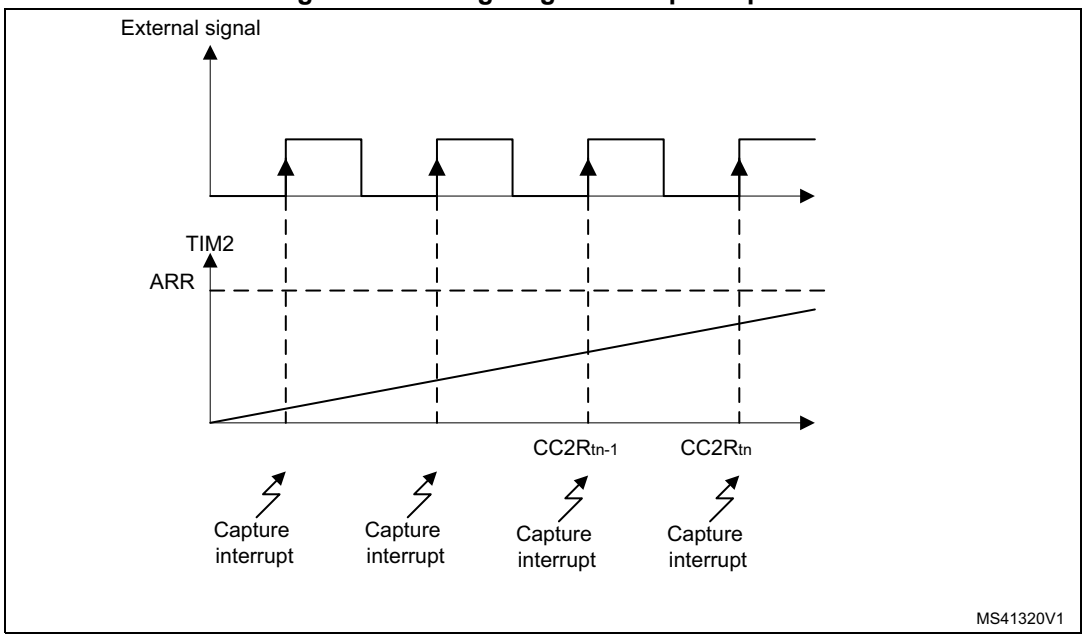
The input capture

To measure the period of an external signal, the timer is used in input capture mode. The maximum frequency, which can be measured with a 32 bit timer, depends on the TIM2CLK signal.

The external signal frequency is not measured directly but it is computed from the number of clock pulses counted using the timer. To do this, a very accurate reference frequency must be available.

After enabling the timer counter, when the first rising edge of the reference signal occurs, the timer counter value is captured and stored in CC2R. It is also stored in a buffer by using the DMA to not overwrite it with the subsequent value at the next rising edge.

Figure 15. Timing diagram of input capture



The elapsed time between two consecutive rising edges $CC2R_{tn}$ - $CC2R_{tn-1}$ represents an entire period of the reference signal.

Since the timer is clocked by the system clock (internal RC oscillator HSI), the real frequency generated by the internal RC oscillator versus the reference signal is given by:

*Measuredfrequency = (CC2R$_{tn}$ - CC2R$_{tn-1}$) * Reference frequency*

The error (in Hz) is computed as the absolute value of the difference between the measured frequency and the typical value.Hence, the frequency error is expressed as:

Error(Hz) = **|** *Measuredfrequency - typicalvalue* |

After calculating the error for each trimming value, the deduction of the ppm is expressed as:

$$ppm = \frac{Error \times 10^6}{typicalvalue}$$

In this example, the timer is clocked by 8 MHz (TIM2CLK = 8 MHz), so the minimum frequency that can be measured without a timer counter overflow is:

$$f = \frac{TIM2CLK}{ARR} = \frac{8 \times 10^6}{0xFFFFFFFF} = 0.002Hz$$

Input signal can be measured from the waveform generator = 1 kHz. The input capture mode is configured as follows:

- The external signal is connected to TIM2 CH2 pin (PA01)
- The rising edge is used as active edge
- The TIM2 CCR2 is used to compute the frequency value.

The input capture module is used to capture the value of the counter after a transition is detected by the input channel 2. To get the external signal period, two consecutive captures are needed. The period is calculated by subtracting these two values.

When the capture event occurs, the CC2IF (register TIM2_SR) is set to 1. If the DMA function is enabled, it generates a DMA request. If the capture occurs, CC2IF flag has been set, then the over sampling flag CC2OF is set.

When a rising edge comes, numerical timer current meter (TIM2_CNT) writes in the TIM2_CCR2. Wait until the next rising edge to, there is be another counter value in TIM2_CNT records. According to the two data value, we can calculate the cycle of input data. Overflow timer is not allowed.

External clock-source mode 2 configuration

The second part of this example is to keep this configuration and change the source of the clock timer to external source mode 2 that uses the ETR pin as timer input clock.

The external signal must meet the synchronization condition:

$$TIM2_{Clk} \geq 3 \times f_{ETRP}$$

In this example the CLK TIM2 = CLK APB1 = 8 MHz, then:

$$f_{ETR} = \frac{1}{3} \times 8\text{Mhz} = 2.66MHz$$

For this, perform a frequency 8 MHz signal waveform generator and use the asynchronous divider to divide by four:

$$f_{ETRP} = 2MHz \leq \frac{1}{3} \times 8MHz$$

TIM2 configuration

```
/* TIM2 Configuration */
/* TIM2 clock enable */
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
/* Set the Timer prescaler to get 8MHz as counter clock */
Prescaler = (uint16_t) (SystemCoreClock / 8000000) - 1; TIM2->SMCR = RESET;
/* Reset the SMCR register */
#ifdef USE_ETR
/* Configure the ETR prescaler = 4 */
TIM2->SMCR |= TIM_ETRPRESCALER_DIV4 |
/* Configure the polarity = Rising Edge */
TIM_ETRPOLARITY_NONINVERTED |
/* Configure the ETR Clock source */
TIM_SMCR_ECE;
#else /* Internal clock source */
/* Configure the Internal Clock source */
TIM2->SMCR &= ~TIM_SMCR_SMS;
#endif /* USE_ETR */
TIM2->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* Select the up counter mode */
```

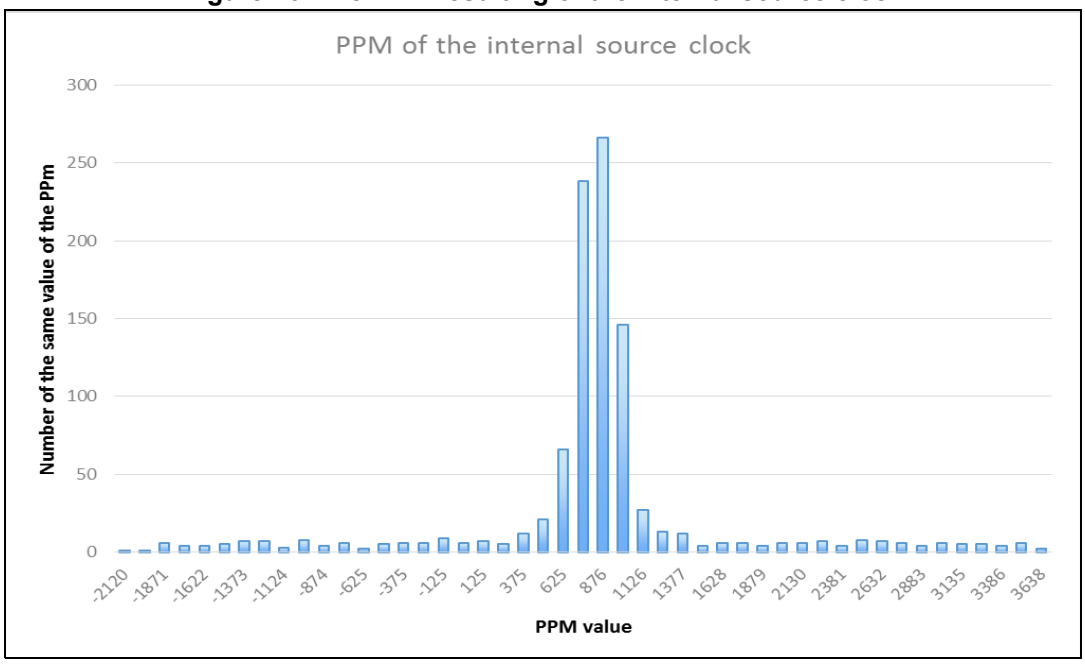
```
TIM2->CR1 |= TIM_COUNTERMODE_UP;
TIM2->CR1 &= ~TIM_CR1_CKD;
/* Set the clock division to 1 */
TIM2->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* Set the Autoreload value */
TIM2->ARR = PERIOD;
/* Set the Prescaler value */
TIM2->PSC = (SystemCoreClock / 8000000)-1;
/* Generate an update event to reload the Prescaler value immediately */
TIM2->EGR = TIM_EGR_UG;
TIM2->CCMR1 &= ~TIM_CCMR1_CC2S;
/* Connect the Timer input to IC2 */
TIM2->CCMR1 |= TIM_CCMR1_CC2S_0;
```

Input capture configuration

```
/* Enable the DMA1 */
DMA1_Channel7->CCR |=  DMA_CCR_EN;
/* Enable the TIM Capture/Compare 2 DMA request */
TIM2->DIER |= TIM_DMA_CC2; /* Enables the TIM Capture Compare Channel 2
TIM2->CCER |= TIM_CCER_CC2E;
/* Enable the TIM2 */
TIM2->CR1 |= TIM_CR1_CEN;
/* wait until the transfer complete */
while ((DMA1->ISR & DMA_ISR_TCIF7) == RESET) {}
```

After taking a 1000 samples and calculate the corresponding PPM, the chart below is obtained.

Figure 16. The PPM resulting of the internal source clock

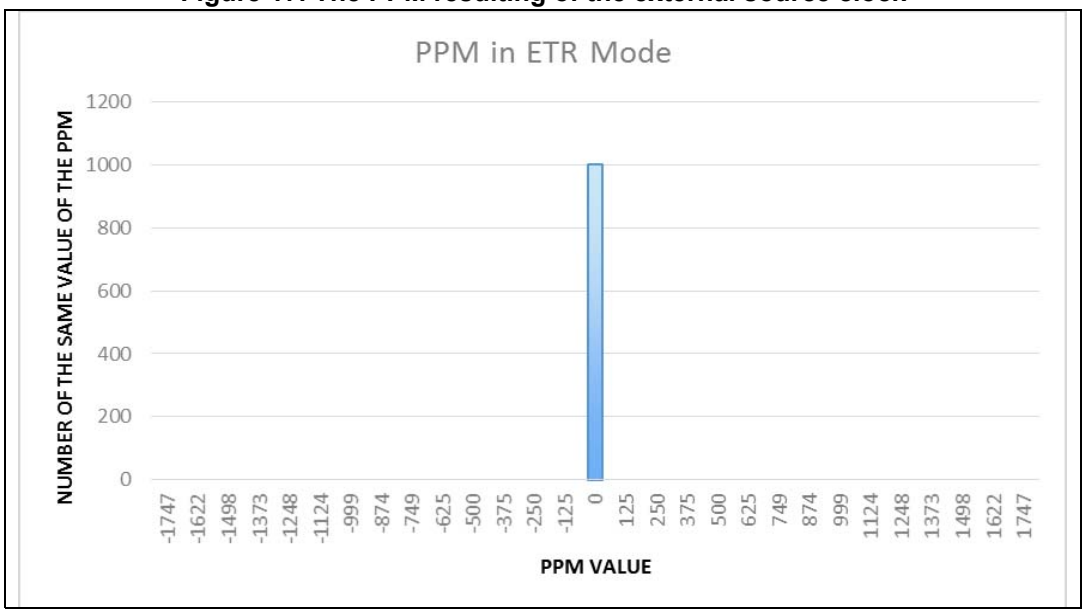


The distribution of PPM values is far from zero and have a static and a dynamic offset. The static offset is due to the instability of the HSI (manufacturing process variations) it is a systematic faults and it varies from one microcontroller to another. The dynamic offset is caused by the environment conditions like the temperature.

Any error resulting from the difference between the actual time base oscillator frequency and the nominal frequency is directly translated into a measurement error. This difference is the cumulative effect of all the individual time base oscillator errors.

For the second part of the application, when the timer is synchronized by the external source clock, the chart below is obtained after taking 1000 samples and calculated the corresponding PPM.

Figure 17. The PPM resulting of the external source clock



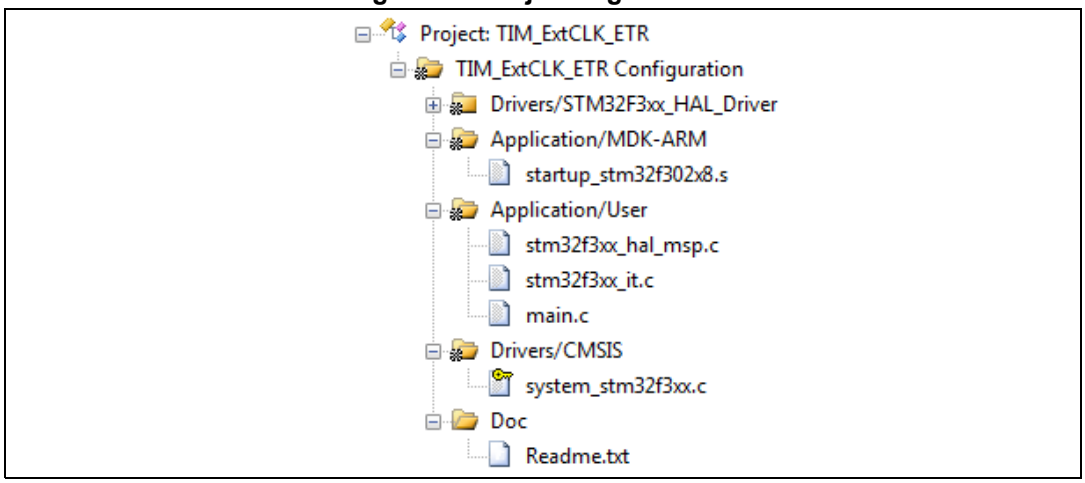
All PPM values are zero because all the time between the rising edges of the measured signal are identical. In this case, note that there are no errors in measurement because the external clock-source is properly calibrated.

2.7 Firmware overview

The firmware was developed on the Keil µvision, IAR Embedded workbench and the SYSTEM WORKBENCH.

The firmware developed is delivered in a ZIP file and contains all the subdirectories and .h and .c source code files that make up the core of the application.

Figure 18. Project organization



The firmware contains all the application task source files and the related files and consists on the following project folders:

- The STM32 MCU HAL library
- The startup file
- The application layer.

3 N-pulse waveform generation using one-pulse mode

3.1 Overview

The one-pulse mode (OPM) of an STM32 timer peripheral is a feature that can be used together with the timer channels configured in output mode. It allows the timer to generate a pulse of a programmable width after a programmable delay on the timer channels configured in PWM1 or PWM2 output compare modes.

This mode is activated by setting the OPM bit in the TIMx_CR1 timer register. Each time that the OPM control bit is set, and a timer update event occurs, the timer counter enable control-bit is reset and the counter is frozen to its value at the update event. In an up-counting configuration, the timer counter register is frozen at 0 value. For other counting configuration such as center-aligned configuration and down-counting mode, refer to the reference manual document to determine the value on which the counter is frozen.

If the timer update event is masked by some means, the one-pulse operating mechanism is unable to reset the CEN control-bit and the timer counter keeps running. The timer's outputs continue to output their configured waveforms.

A possibility to mask the effect of a timer update event, is to set the UDIS control bit. For details on this control bit, refer to the reference manual document. Another way to mask the timer update event is by making the timer repetition counter register's content different from zero. Not all the STM32 timer peripherals embed the repetition counter.

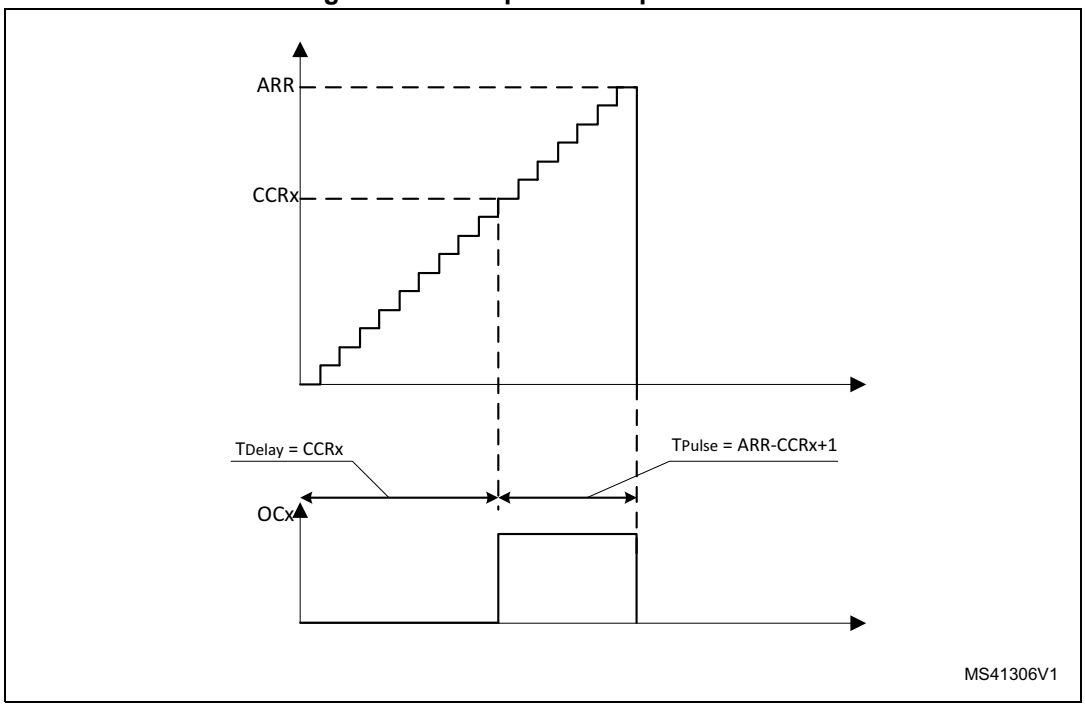
Given the previous condition, and considering that each time that the update event occurs (with masked effect), the repetition counter content is decremented, then, this behavior can be used to generate a series of pulses.

For instance, to generate a series of five pluses, the repetition counter should be set to four. The first four updated events are masked as the repetition counter content is different from zero. These four first update events are not reset the CEN control bit but they decrement the repetition counter content down to zero. The fifth update event is the one that resets the CEN control (the repetition counter is already at zero). This way, the timer counter has overflowed five times before being disabled and the five required pulses are generated.

With default channel output polarity, the PWM2 output mode gives the typical pulse waveform, like a delay for a given time, then a pulse with given duration. For inversed polarity, either the channel output polarity should be inversed or the PWM1 output mode should be used.

The first scenario with a default channel-output polarity is taken as example below in *Figure 19*. The outputted waveform is characterized by two parameters: the pulse and the delay lengths.

Figure 19. Example of one-pulse mode



The T_{Delay} is defined by the value written into the TIM_CCRx timer capture/compare register.

The T_{Pulse} is defined by the difference between the value written into the timer auto-reload register (TIMx_ARR) and the T_{Delay} value. The value of TIMx_ARR – value of TIMx_CCRx + 1.

3.2 Application: N-pulse waveform generation using one-pulse mode

The objective of this application is to generate a waveform composed of a series of defined number of pulse in one timer channel output. This is achieved using the one-pulse mode and the repletion counter features of the STM32 timer peripheral.

The TIM1 timer peripheral is selected for this application because it meets the needed requirements.

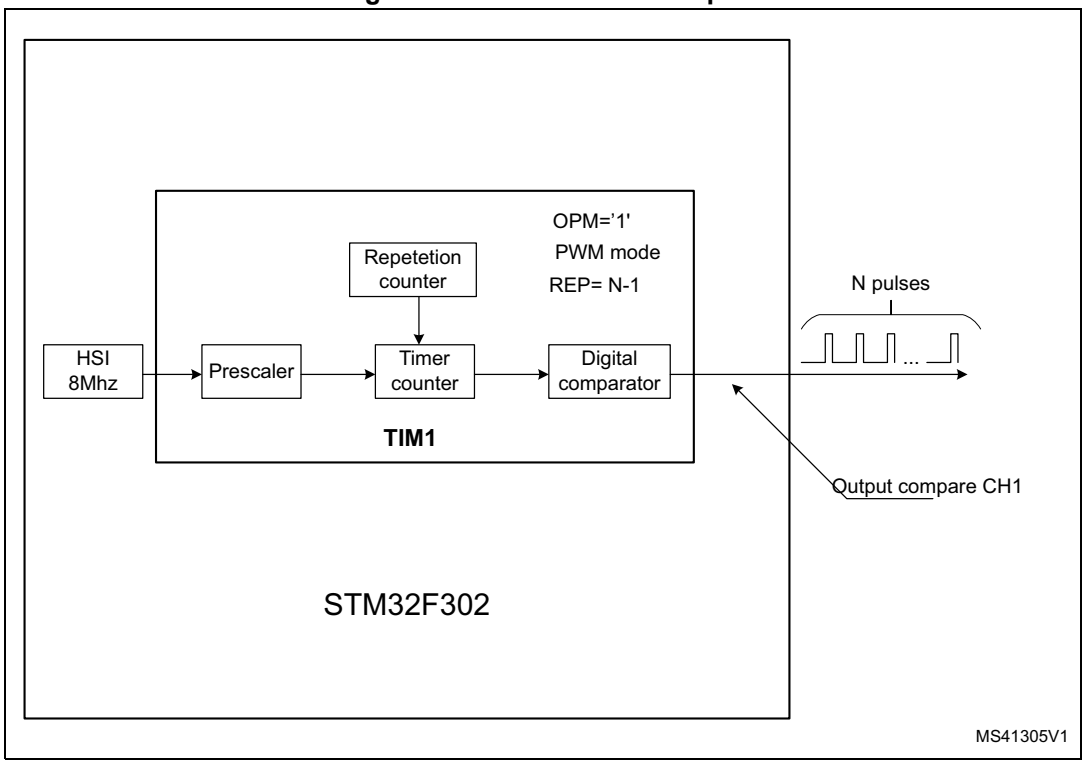
To run this application the following hardware set-up is needed:

- An STM32F302 Nucleo board (NUCLEO-F302R8).
- An USB cable to connect the Nucleo board to the development PC (the USB cable also provides the power supply for the board).
- An oscilloscope to visualize the output signal waveform.

To reshape this application source code and debug different configurations, a set of software tools are needed:

- A development tool-chain supporting the ST-LINK debugger
- The STM32CubeMX software tool or later versions (v4.10.1 was the latest release when this application was developed).

Figure 20. Architecture example



In this application, the timer is clocked by the internal HSI oscillator. Set the OPM bit in the TIMx_CR1 timer register to activate the OPM mode. To obtain the N-pulses at the outputs, the right value (N-1), must be configured into the TIMx_RCR register.

System clock initialization

Clock initialization is done in the main routine using the SystemClock_Config() function right after the HAL library initialization (HAL_Init).

The configuration code is as below:

N-pulse waveform generation using one-pulse mode configuration

```
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* Peripheral clock enable */
Prescaler = (uint16_t) (SystemCoreClock / 1000000) - 1;
/* Set the Timer prescaler to get 1MHz as counter clock */
TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);  /* Select the up counter mode */
TIM1->CR1 |= TIM_COUNTERMODE_UP;
TIM1->CR1 &= ~TIM_CR1_CKD
TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1; /* Set the clock division to 1*/
TIM1->ARR = PERIOD; /* Set the Autoreload value */
TIM1->CCR1 = PULSE; /* Set the Pulse value */
TIM1->PSC = Prescaler; /* Set the Prescaler value */
TIM1->RCR = PULSE_NUMBER - 1; /* Set the Repetition counter value */
TIM1->EGR = TIM_EGR_UG; /* Generate an update event to reload the Prescaler
and the repetition counter value immediately */
TIM1->SMCR =  RESET;  /* Configure the Internal Clock source */
```

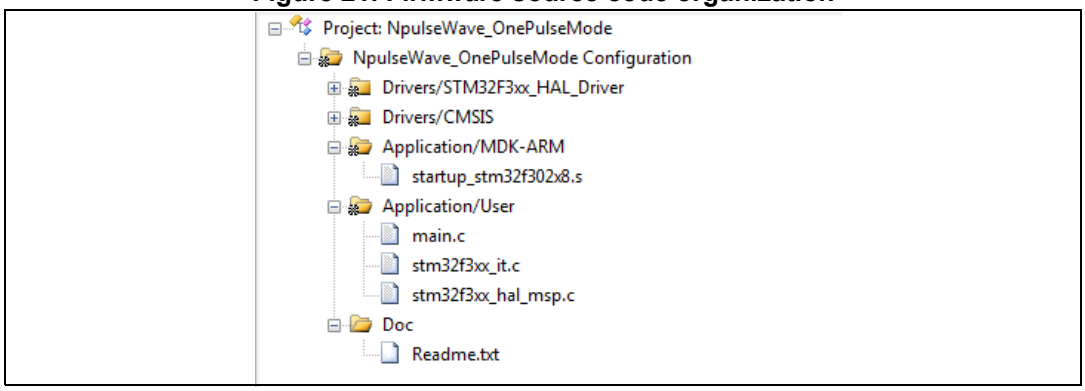
```
TIM1->CR1 |= TIM_CR1_OPM; /* Select the OPM Mode */
TIM1->CCMR1 &= (uint16_t)~TIM_CCMR1_OC1M;
TIM1->CCMR1 &= (uint16_t)~TIM_CCMR1_CC1S;
TIM1->CCMR1 |=  TIM_OCMODE_PWM2;
/* Select the Channel 1 Output Compare and the Mode */
TIM1->CCER &= (uint16_t)~TIM_CCER_CC1P;
/* Set the Output Compare Polarity to High */
TIM1->CCER |= TIM_OCPOLARITY_HIGH;
TIM1->CCER = TIM_CCER_CC1E; /* Enable the Compare output channel 1 */
TIM1->BDTR |= TIM_BDTR_MOE; /* Enable the TIM main Output */
TIM1->CR1 |= TIM_CR1_CEN; /* Enable the TIM peripheral */
```

3.3 Firmware overview

The firmware was developed on the Keil µvision, IAR Embedded workbench and the SYSTEM WORKBENCH.

The firmware developed is delivered in a ZIP file and contains all the subdirectories and “.h” and “.c” source code files that make up the core of the application.

Figure 21. Firmware source code organization



The firmware contains all the application task source files and the related files and consists on the following project folders:

- The STM32 MCU HAL library
- The startup file
- The application layer.

4 Cycle-by-cycle regulation using break input

4.1 Overview

The Break feature is available in the advanced-configuration timer peripherals such as TIM1 and TIM8 timers, as well as in the lite-configuration timer peripherals like TIM15, TIM16 and TIM17 timers. The Break feature is mainly used for protecting the power stage driven by the timer outputs; it disables or forces them into a predefined safe state as soon as something goes wrong within the power stage or within the microcontroller itself.

To detect external break requests generated by the power stage, the break function is associated with a dedicated input (for example BKIN or BKIN2), which is mapped as an alternate function on several IO of the microcontroller. Once enabled, the break function disable the PWM outputs or forces them to a predefined safe state as soon a break event is detected, even if no clock is present. For example, it is possible to have an asynchronous deactivation of outputs.

The reference manual document contains more information about how to configure the safe predefined states for timer outputs.

As soon as a valid break-event is detected, the MOE control-bit in the TIMx_BDTR register is cleared asynchronously. It acts only on timer channels configured in output mode. In order to resume the normal operating state of the timer channel outputs, the MOE control bit should be set by software or the AOE control bit should be set in order for the timer outputs to resume their normal operating state by the start of a new PWM cycle.

In some other STM32 microcontrollers, such as STM32C5 and STM32G5, it is possible to override the output states when they are disabled (i.e. MOE bit reset in the TIMx_BDTR register).

This feature can be used to apply specific patterns before starting PWM operations, for instance to refresh bootstrap capacitors of power-switch gate drivers.

The OOC control bit in the TIMx_OOR register acts as follows:

- OOC = 0: tim_ocx and tim_ocxn outputs are in the default idle state.
- OOC = 1: tim_ocx and tim_ocxn outputs are forced to an idle state defined by the OOSx/OOSxN bits in the TIMx_OOR register.

Some of the STM32 microcontroller families (for example the STM32F303 family) embed timer peripherals that feature two external break inputs: BKIN and BKIN2. For those timer peripherals, the BRK2 break input has a lower priority than the BRK break input. The BKIN and BKIN2 break inputs are able to disable the timer channel outputs only when forcing them to Hi-Z state, and they are not able to force them to a predefined safe state.

4.2 Break input versus OCxRef-clear utilization

As mentioned on previous section, the main use of the break input is to manage the timer channel outputs in faulty conditions. Thanks to its flexible design, the break input can be also used in other uses cases like the cycle-by-cycle current regulation. Conceptually, the cycle-by-cycle current regulation function is fulfilled by the OCxRef-clear feature of the STM32 peripheral timers.

In some situations it is impossible to use the OCxRef-clear feature to manage the cycle-by-cycle current regulation function. This is due to the fact that the ETR timer input used by this

feature may be concurrent with another timer feature (for example the timer external synchronization uses the ETR timer input for the external clock). In such situation it may be useful to use the Break feature to handle the cycle-by-cycle current regulation function.

In one hand, the OCxRef clear feature acts on the timer channel outputs (which is also the case for the Break feature); on the other hand, the OCxRef feature can be activated per channel (for example the activation of the OCxRef clear feature is channel-wise through the setting of the respective OCxCE control bit for the regulated channels). In this secod situation, the Break feature is acting on all the timer channel outputs (for example no means to control which channel is affected by the break event).

The concept of cycle-by-cycle current regulation consists on the fact that as soon as the regulated current magnitude is above a predefined threshold, the PWM signal be turned low until the beginning of the next PWM cycle. This behavior is supported natively by the OCxREf-clear feature and is illustrated in *Figure 22*.

For the Break feature, this behavior is controlled by the AOE control bit. If the AOE control bit is kept in its default state (reset state) then as soon as the regulated current crosses the predefined threshold, the timer outputs become low. The timer continue to be in that state until the MOE bit is set by the software which is not the typical behavior of a cycle-by-cycle regulation logic.

Setting the AOE control bit before activating the regulation function through the Break feature sets the MOE control bit automatically at the start of each new PWM cycle. The previous described behavior of the Break feature versus the AOE control bit configuration is illustrated in *Figure 23* where the waveform on one timer channel output is plotted.

Figure 22. Timing of clearing TIMx OCxREF

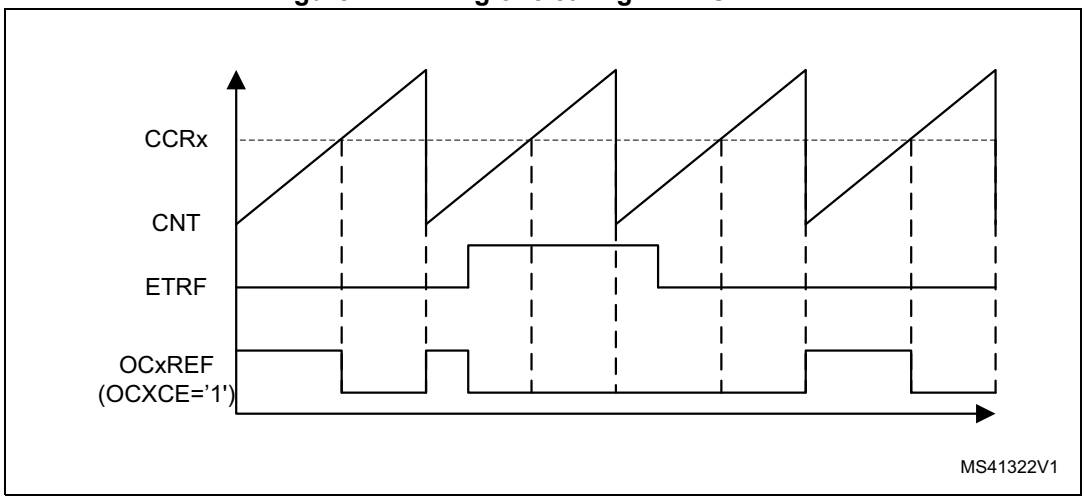


Figure 23. Timing of Break function

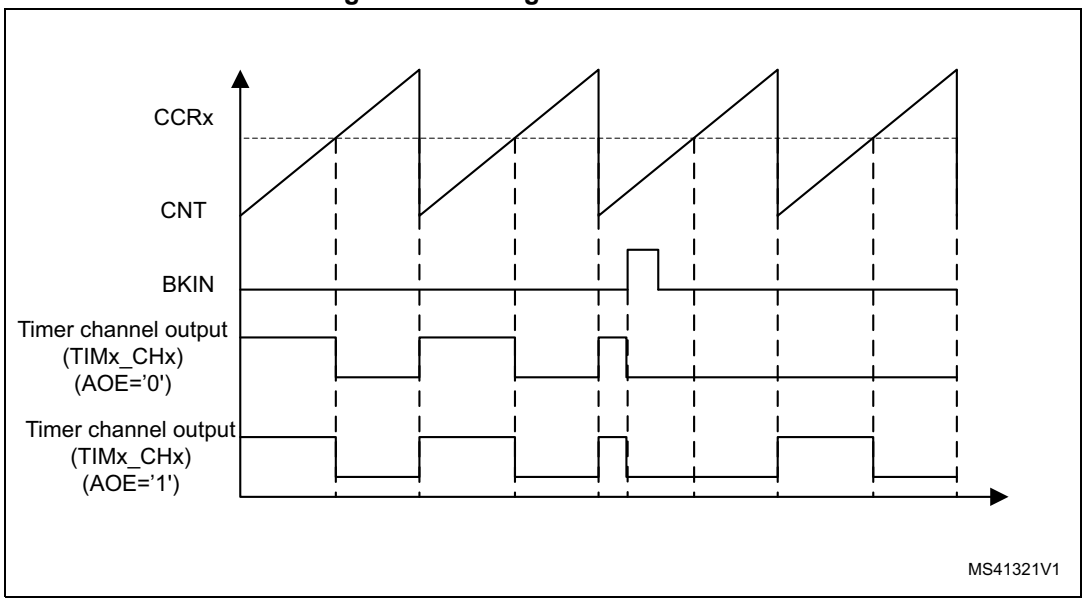


Figure 23 illustrates the different behaviors of the timer's output between the two configurations of the AOE control bit.

- If AOE = '0', the PWM output is disabled even if the break input is not active
- If AOE = '1', the PWM output is enabled at the next update event if the break input is not active.

4.3 Application: cycle-by-cycle regulation using the Break feature

The cycle-by-cycle regulation is one of the employed techniques to implement an over-current protection at the level of power-conversion stages. This function monitors the current that is drawn by the power stage. As soon as the current overpasses a predefined threshold (I_{Ref}), the PWM signal is turned low to make the drawn current decrease below the threshold in a regulation effect.

The PWM signal is not allowed to resume its configured duty-cycle until the beginning of a new PWM cycle. If the current drawn is still higher than the threshold, the PWM signal continue to be at low state until the next PWM cycle and so on.

The comparison of the current drawn by the power stage with the predefined threshold current may be realized with a comparator-based electronic circuit. The comparator itself can be embedded by the microcontroller; (for example this is the case for the STM32F302 and STM32F303 microcontrollers). The output of that comparator-based circuit should be fed into the break input and respectively the ETR input in the case where the OCxRef-clear feature is used.

Figure 24. Timing of cycle-by-cycle regulation

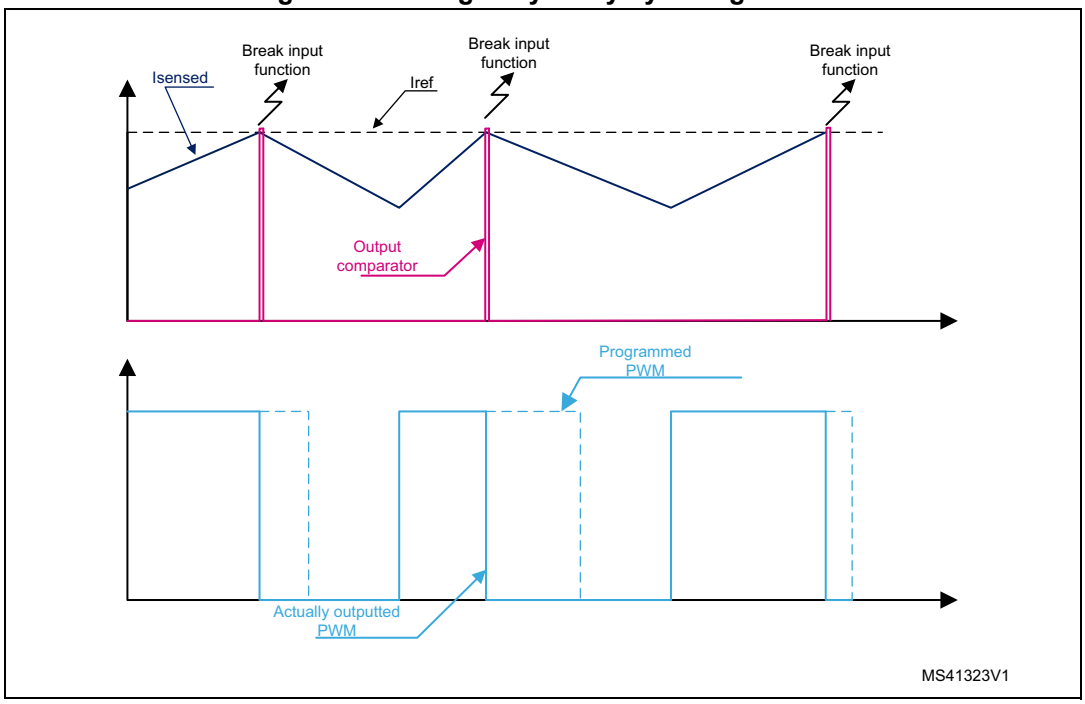
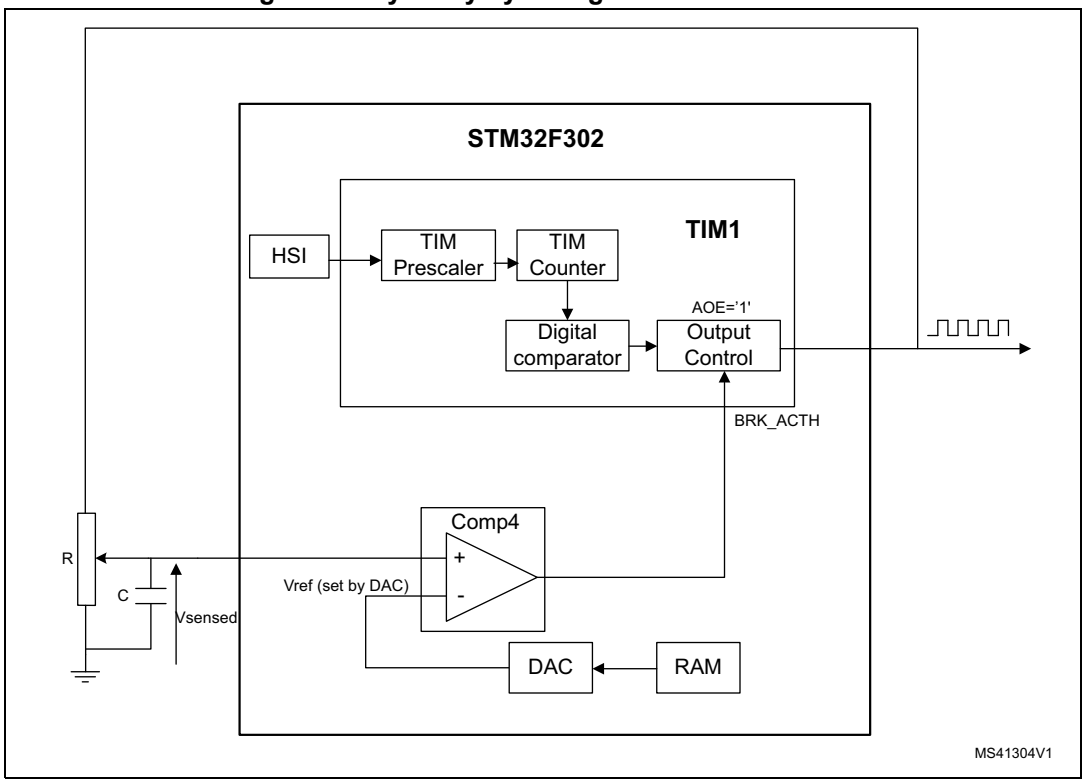


Figure 24 shows the behavior of the break function when the break is enabled. It shows both the actually outputted PWM (solid line) and the programmed PWM (dotted line) value.

The break function drives the output channel low by applying a high level to the BRK_ACTH input. The break function remains low until the next “update event” because the AOE bit control is set.

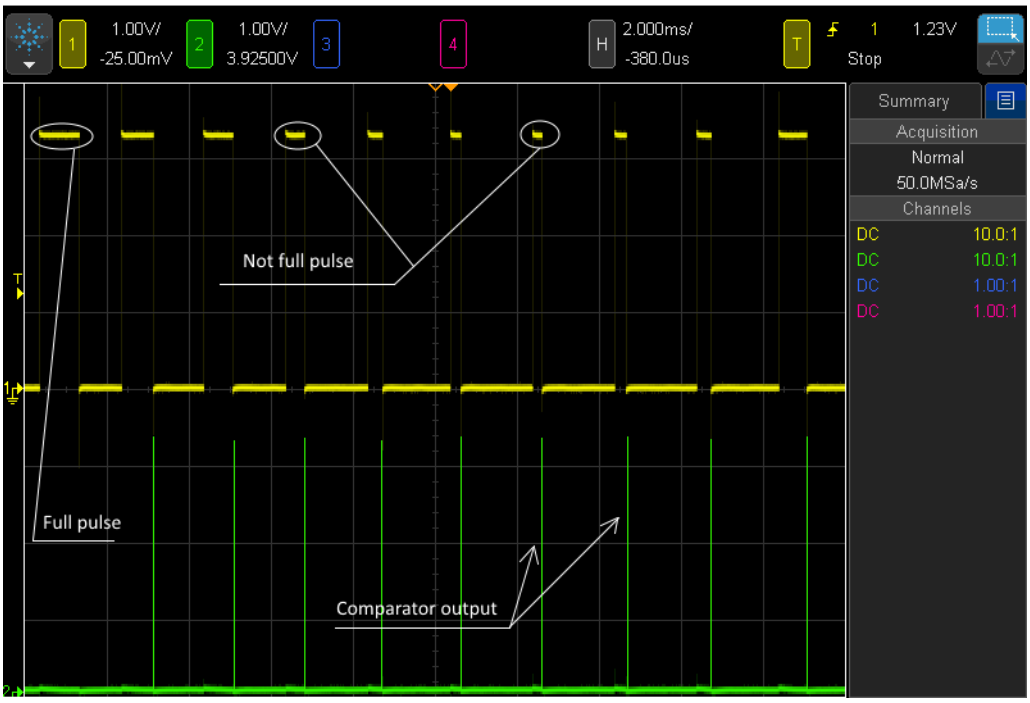
Figure 25 shows an emulation of the cycle-by-cycle regulation concept using the Break feature. A simple electronic circuit made of a passive electronic components (resistor and capacitor) was used to emulate the feedback signals outputted by a real regulated power inversion system.

Figure 25. Cycle-by-cycle regulation architecture



As displayed in *Figure 25*, the timer is used in PWM mode, the Vref is stocked in RAM and it is compared with the input voltage after conversion with the DAC and the analog comparator. The comparator output is internally redirected to the BRK_ACTH signal which controls the PWM state.

Figure 26. Oscilloscope screen-shot for the obtained waveform



Cycle-by-cycle regulation is one of the employed techniques to implement an over-current protection at the level of power conversion stages. *Figure 26* shows the variation of the duty-cycle value in function of time due to the variation of the input voltage relative to Vref.

In this figure, during the first PWM pulse the voltage reaches the 'Vref limit' value before the PWM input pulse ends. The PWM output pulse is shut off early in its cycle. Then the tension decays until the moment when the next pulse is applied, which once again causes the tension to rise. During the second PWM pulse no current limit occurs because the pulse ends before the rising current reaches the 'current limit' threshold.

To run this application, the following hardware set-up is needed:

- An STM32F302 Nucleo board (NUCLEO-F302R8)
- An USB cable to connect the nucleo board to the development PC (the USB cable also provides the power supply for the board).
- An oscilloscope to visualize the output signal waveform
- A potentiometer and a capacitor

To reshape this application source code and debug a different configuration, a set of software tools are needed:

- A development tool-chain supporting the ST-LINK debugger
- The STM32CubeMX V4.10.1 software tool or later versions (v4.10.1 was the latest release when this application was developed).

The system-clock initialization

Clock initialization is done in the main routine using the SystemClock_Config() function right after the HAL library initialization (HAL_Init).

Break-function configuration

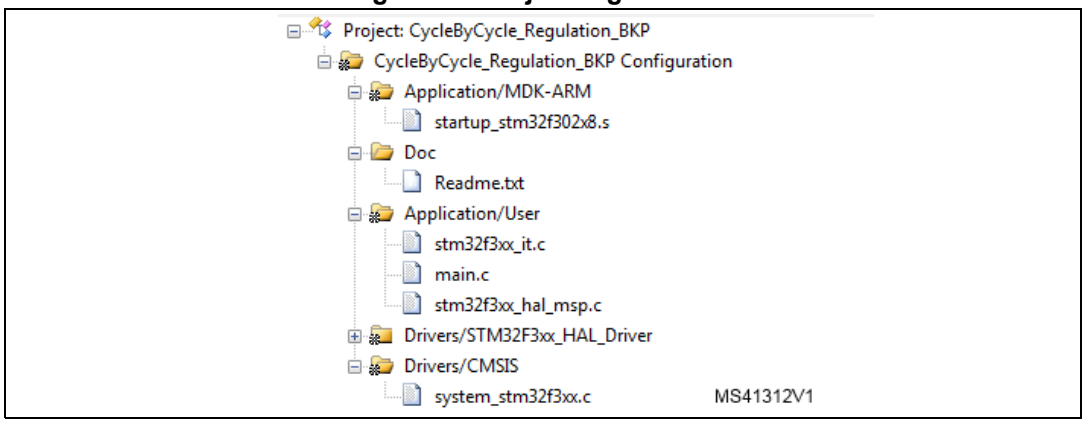
```
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* TIM1 clock enable */
Prescaler = (uint16_t) (SystemCoreClock / 500000) - 1; /* Set the Timer
prescaler to get 500 kHz as counter clock */
TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);  /* Select the up counter mode*/
TIM1->CR1 |= TIM_COUNTERMODE_UP;
TIM1->CR1 &= ~TIM_CR1_CKD;
TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1; /* Set the clock division to 1*/
TIM1->ARR = PERIOD;  /* Set the Autoreload value */
TIM1->CCR1 = PULSE;  /* Set the Capture Compare Register value */
TIM1->PSC = Prescaler; /* Set the Prescaler value */
TIM1->EGR = TIM_EGR_UG; /* Generate an update event to reload the Prescaler
and the repetition counter value immediatly */
TIM1->SMCR = RESET;  /*Configure the Internal Clock source */
TIM1->BDTR = RESET; /* Clear the BDTR bits */
TIM1->BDTR |= DEAD_TIME; /* Set the Dead Time value to 0 */
TIM1->BDTR |= TIM_LOCKLEVEL_OFF;  /* Disable the Lock Level*/
TIM1->BDTR |= TIM_OSSI_ENABLE;  /* Enable the Output idle mode */
TIM1->BDTR |= TIM_OSSR_DISABLE;  /* Disable the Output run mode */
TIM1->BDTR |= TIM_BREAK_ENABLE; /* Enable the Break input */
TIM1->BDTR |= TIM_BREAKPOLARITY_HIGH; /* Set the polarity to High */
TIM1->BDTR |= TIM_AUTOMATICOUTPUT_ENABLE; /* Enable the automatic output */
TIM1->CCMR1 &= ~TIM_CCMR1_OC1M; /* Select channel 1 output Compare and Mode
*/
TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
TIM1->CCMR1 |= TIM_OCMODE_PWM1;
TIM1->CCER &= ~TIM_CCER_CC1P; /* Set the Output Compare Polarity to High */
TIM1->CCER |= TIM_OCPOLARITY_HIGH;
TIM1->CCER |= TIM_CCER_CC1E; /* Enable the Compare output channel 1 */
TIM1->CR1|=TIM_CR1_CEN; /* Enable the TIM peripheral */
```

4.4 Firmware overview

The firmware was developed on the Keil µvision, IAR Embedded workbench and the SYSTEM WORKBENCH.

The firmware developed is delivered in a ZIP file and contains all the subdirectories and .h and .c source code files that make up the core of the application.

Figure 27. Project organization



The firmware contains all the application task source files and the related files and consists on the following project folders:

- The STM32 MCU HAL library
- The startup file
- The application layer.

5 Arbitrary waveform generation using timer DMA-burst feature

5.1 STM32 DMA-burst feature overview

The direct memory access (DMA) peripheral is used in order to provide a high-speed data transfer both between peripherals and memory and also between memory and memory. This action saves CPU resources that could be used in other tasks.

Each DMA transfer is made of two stages:

- In the first stage, the data to be transferred is loaded from the source location
- In the second stage, the retrieved data is stored into the destination location.

This two-stage data transfer operation is associated with an update of the DMA peripherals transfer index register; this is the register used to track how much data is left to be transferred.

In STM32 microcontroller families, there are two DMA peripheral variants:

- The DMA burst transfer feature supported only by the STM32F2 products DMA peripheral variant where the DMA peripheral can transfer a configurable number of data elements next to a single data transfer trigger.
- In another variant, like the one on the STM32F1 products, the DMA peripheral variant supports only single transfers; this means that only one data element is transferred next to a data transfer trigger.

The paragraph above is not intended to give a detailed description of the STM32 DMA-burst feature supported by many STM32 microcontrollers. The information stated above aims to mitigate any misunderstanding of this feature and any possible confusion with the STM32 timer burst feature which makes the focus of this chapter.

For more information about the STM32 DMA peripheral, consult the STM32 microcontrollers documentation reference manuals and application note *STM32 cross-series timer overview* (AN4013).

5.2 Timer DMA-burst feature

The timer peripherals have the capability to generate multiple successive DMA requests next to a single timer event. The main usage of this feature is to update the content of multiple registers of the timer peripheral each time a given timer event is triggered. This can be done either to dynamically reconfigure the timer operating mode (switching from one output mode to another, for instance from PWM2 mode to force-active-level mode), or to change the run-time parameters on several channels simultaneously (change the duty-cycles for more than one timer channel at once).

The same feature can also be used to transfer the content of a number of timer peripheral registers out into a memory buffer.

This feature allows to modify on the fly the waveform outputted by a timer peripheral output by adjusting the timer peripheral registers content. For example, it allows to update the TIMx_ARR register content to adjust the outputted waveform frequency or update the TIMx_CCRx register to adjust the signal duty-cycle.

In order to use the timer's DMA-burst feature the programmer has to deal with below timer peripheral registers:

- The DMA address register (TIMx_DMAR): this is the read/write access redirection register.
- The DMA control register (TIMx_DCR): this is the burst-transfer state-machine control register
- DMA controller's peripheral-address-register setting.

Timer peripheral DMA-burst feature address register

The peripheral address register, within the DMA peripheral, is used to configure the transfer destination register address when the DMA is configured in memory-to-peripheral mode. It is also used to configure the transfer source register address when the DMA is configured in peripheral-to-memory mode.

The DMA peripheral should update the content of a series of timer peripheral registers with the content of a memory buffer of predefined and/or precomputed values, but it does not mean that the transfer address pointed by the DMA peripheral address register should be configured to be post-incremented by the DMA controller next to each data transfer. The DMA controller peripheral address register has to point to the TIMx_DMAR timer peripheral register.

The TIMx_DMAR timer register is a virtual register. Any access to this register is redirected by the timer peripheral DMA-burst control logic to one of the timer peripheral physical registers.

Access to the TIMx_DMAR register can be either of read or write type. The redirection of an actual access to the TIMx_DMAR register to another timer peripheral physical register depends on the content of the TIMx_DCR register on the DMA-burst interface settings. It depends also on the actual state of the finite-state machine (FSM) controlling the timer DMA burst interface.

DMA controller's memory-address-register setting

The memory address register within the DMA peripheral is used to configure the transfer destination memory location address when the DMA is configured in peripheral-to-memory mode. It is used also to configure the transfer source memory location address when the DMA is configured in memory-to-peripheral mode.

The DMA peripheral should update the content of a series of timer peripheral registers with the content of a memory buffer. The transfer address pointed by the DMA memory address register should be configured to be post-incremented by the DMA controller next to each data transfer.

Timer peripheral DMA-burst feature control register

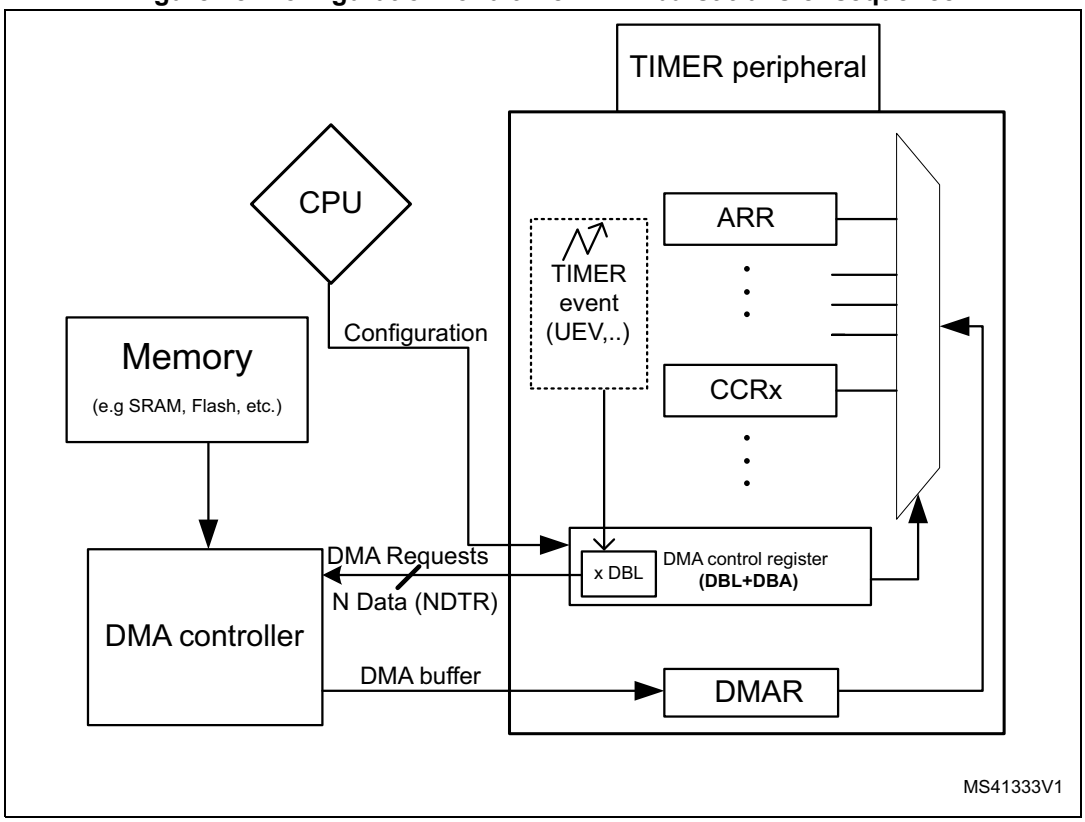
The state-machine control register for the timer DMA burst feature, TIMx_DCR, is used to configure the number of beats during a single burst transfer. It is also used to identify the target timer register or the source of the first data transfer during a burst transfer.

The DBL [4:0] control bit-field sets the number of beats during one burst transfer that should be equal to the number of timer registers that is involved (either write or read) in a burst transfer. The content of the DBA [4:0] control bit-field identifies the start register involved in a burst transfer among the timer registers.

The DBA[4:0] control bit-field can identify up to 32 timer registers as it is of 5-bit width. Timer registers identification number is obtained from dividing the register relative address within the timer register map by four. For example, the TIMx_CR1 register has a relative address of 0x00, then its identification number is 0.

To make a burst transfer start from TIMx_CR1 register, the DBA[4:0] control bit-field should be set to 0. To make the burst transfer start from the TIMx_ARR register, the DBA[4:0] bit-field should be set to 11.This means 44 / 4 = 11 where 44 is the relative address of the TIMx_ARR register, 0x2C, after being encoded into decimal base.

Figure 28. Configuration for a timer DMA-burst transfer sequence



In order to complete one timer DMA-burst transfer sequence, as shown by *Figure 28*, the DMA and the timer peripherals should cooperate. The data values that used to update the timer registers should be stored somewhere within the microcontroller memory. They can be stored in the SRAM memory if the pattern should be updated during the waveform generation.

The application software should configure the DMA to point to the data buffer as the source of the data transfer; it should point to the timer TIMx_DMAR register as the destination of the data transfer. Finally, the application software should configure the timer DMA-burst feature by writing the correct settings into the timer TIMx_DCR register.

The above paragraphs provide a detailed description on how to configure the timer DMA burst feature. As soon as the right configuration is done, the DMA stream or channel used for the data transfer should be enabled, followed by the enable of the timer counter.

Once enabled, the timer counter is periodically updated by being either incremented or decremented. After a while and depending on the enabled timer DMA requests, the timer peripheral may raise a DMA request internally. The DMA request is routed internally to the timer DMA-burst control logic.

Based on the value configured into the DMA-burst length (DBL[4:0], bit-filed within the TIMx_DCR timer register), the DMA request is sent out into the DMA peripheral either as is or multiplied several times.If the DBL[4:0] content is null then the DMA request is sent as is; else the DMA request is multiplied by DBL value + 1 factor.

If the DBL[4:0] bit-filed content is 2, then, as soon as one timer DMA request is raised internally, the timer DMA burst control logic send out a first DMA request into the DMA peripheral. The DMA peripheral transfers the content of the memory location; this location is pointed by the DMA register transfer source into the TIMx_DMAR timer register. It then increments the source pointer and acknowledge the timer DMA request.

As soon as the first DMA acknowledgment is received, the timer DMA-burst control logic sends out a second DMA request. This DMA request is handled again by the DMA peripheral and an acknowledgment is sent again to the timer.

After receiving the second DMA acknowledgment, the timer DMA-burst control logic sends out a third DMA request. This third request is again handled by the DMA peripheral.

As soon as the timer receives the third acknowledgment from the DMA, the transfer sequence triggered by the internally raised DMA request is considered as successfully terminated. Then the timer DMA-burst control logic is ready for a new transfer sequence.

For the previous described data transfer sequence, the transfer destination register pointed to by the DMA peripheral remains the same along the whole transfer sequence. The register remains then equal to the timer TIMx_DMAR register.

The timer DMA-burst control logic redirects, each time, the write access into the timer TIMx_DMAR register to the right physical timer register.

For the previous example, and if the DMA base address bit-field DBA[4:0] within the TIMx_DCR register is set to 11, note that:

- The first DMA write access to the timer TIMx_DMAR register is redirected to the TIMx_ARR timer register.The TIMx_ARR timer register is chosen to be the base address for the burst transfer.
- The second DMA access to the TIMx_DMAR timer register is redirected to the TIMx_RCR timer register (assuming that the timer used for this example embeds the TIMx_RCR register).
- The third DMA access to the TIMx_DMAR timer register is redirected to the TIMx_CCR1 timer register.
- At the end of the third DMA access to the TIMx_DMAR register, the timer DMA-burst control logic loops back the write access redirection state-machine. It points again to the configured base register for the burst transfer (the TIMx_ARR timer register in this example).

For each new DMA request generated internally within the timer peripheral, the sequence described above is repeated.

To use the timer DMA burst to read out the content of timer peripheral registers periodically into a certain memory buffer, the above sequence is also valid. Only the DMA transfer direction, source address and destination address should be modified.

5.3 Application example: arbitrary waveform generation using timer DMA-burst feature

This example demonstrates one possible usage of the STM32 timer peripherals to generate an arbitrary waveform signal without CPU overhead. This example aims also to demystify the DMA-burst timer feature which is the keystone for this example.

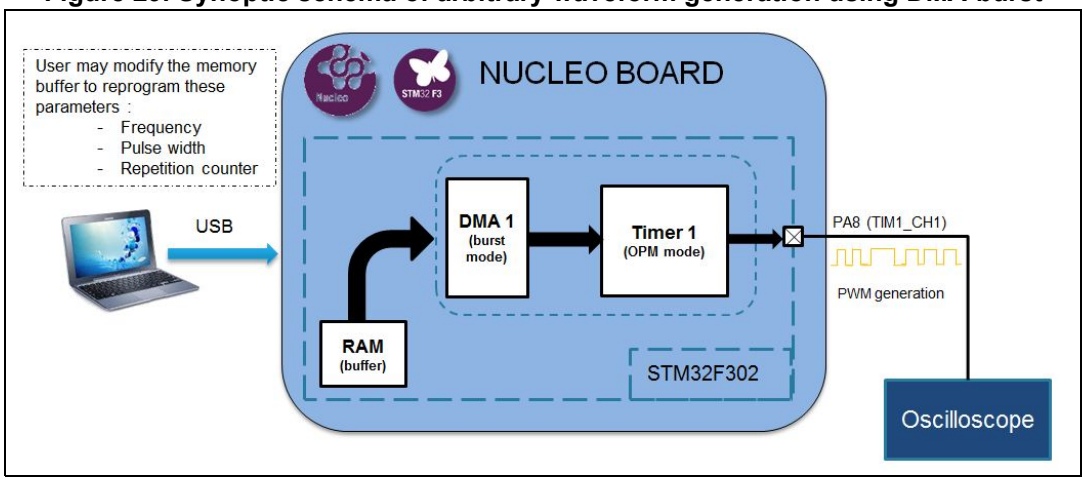
Application overview

This application is designed around the Nucleo board NUCLEO-F302R8, based on STM32F302x8 microcontroller, as the board main controller. Nevertheless, this application can be ported easily to any STM32 hardware platform.

The Nucleo board embeds its own ST-Link/V2 debugger. Only a USB cable is needed to interface the Nucleo board with a PC computer both for application-binary-image download into the microcontroller and for the application debug purposes.

Figure 29 illustrates the synoptic diagram for the arbitrary waveform generator application presented by this application example. The signal generated is outputted on the PA.08 IO of the STM32F302 microcontroller which is mapped to the pin 8 of the connector CN9 of the Nucleo board.

Figure 29. Synoptic schema of arbitrary waveform generation using DMA-burst

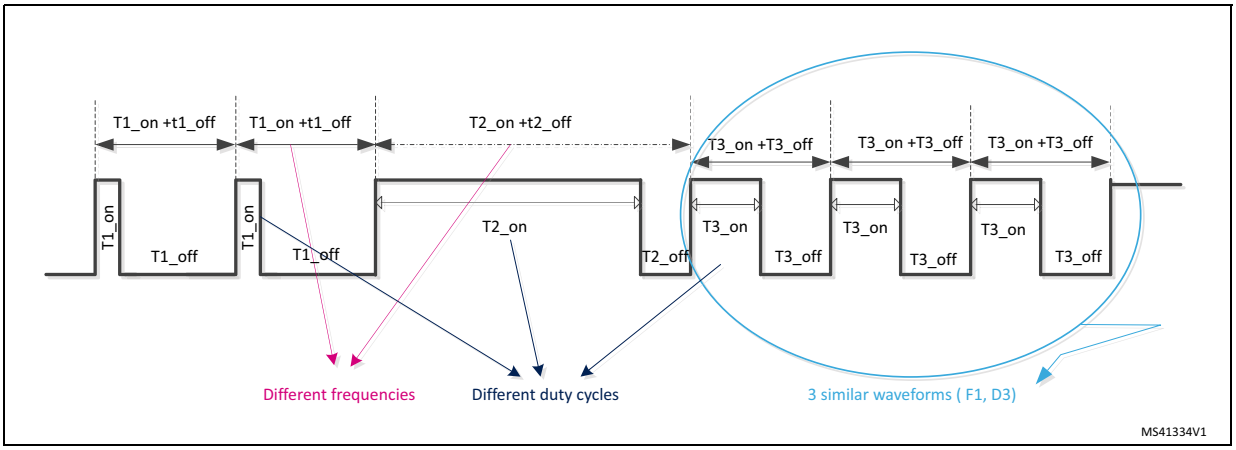


The arbitrary waveform generator described by this example is designed to output a waveform similar to the one shown by *Figure 30*. The source code of this application can be easily reshaped to output different waveforms, but for demonstration purposes the one illustrated in *Figure 4 on page 15* is targeted.

The targeted waveform in *Figure 30* is composed by three waveform portions:

- The first portion is made by two consecutive pulses with a certain on-time and off-time: t1_on and t1_off.
- The second portion is made by one pulse with a different on-time and off-time: t2_on and t2_off.
- The third portion is made by three pulses with a third set of on-time and off-time parameters: t3_on and t3_off.

Figure 30. Arbitrary waveform generator application: targeted waveform



To make an STM32 timer peripheral output this waveform, a timer peripheral featuring a repetition counter can be used. The timer peripheral embeds a TIMx_RCR register and features at least one timer channel. For this example, the TIM1 timer peripheral is used as It features four timer channels as well as a repetition counter.

A timer channel can generate a PWM signal with a constant predefined frequency and duty-cycle parameters. The signal frequency parameter is controlled by the content of the timer TIMx_ARR auto-reload register. In this register, the duty-cycle parameter is controlled by the content of the timer channel 1 register (TIM1_CCR1). In order to make the timer channel output the requested waveform (refer to *Figure 4*), the content of these two timer register at the end of each PWM cycle must be updated.

In other words, it is required to update the content of these two registers synchronously with each timer “update event”. The intuitive way to do this is by making the application software (the CPU unit) update the content of these two registers each time a timer “update event” is raised. Intuitively, these overheads the CPU unit and make the outputted waveform not deterministic.

The following paragraphs explain how to use some of the timer features in conjunction with the DMA peripheral to mitigate the CPU overhead and to generate a deterministic waveform signal.

To update the TIMx_ARR and TIMx_CCR1 timer registers simultaneously next to one timer “update event”, the use should use the DMA-burst timer feature in addition to one DMA channel or stream. How to configure and use the timer DMA-burst feature is described in previous sections of this document.

To prevent the timer channel from generating unwanted glitches due to updating the timer channel register, the preload feature of the STM32 timer peripherals is used.

To make the timer channel to output repetitive pulses with the same t_on and t_off parameters and without the timer generating an “update event” on each PWM cycle, the timer repetition counter should be used.

In order to output the waveform illustrated in *Figure 29* on channel 1 of TIM1 timer peripheral, the content of the below three registers should be updated as follow:

- **TIM1_ARR:** contains the sum of t_on and t_off periods for the on-going pulse.
- **TM1_RCR:** contains the number of pulses per the on-going signal waveform portion minus one (i.e.TIM1_RCR=Number_of_pulses_per_portion–1).
- **TIM1_CCR1:** contains the duration of the t_on period of the on-going pulse.

The configuration for the timer registers to be used in order to reconstitute the waveform shown in *Figure 29* are the following:

For the first portion of the waveform, the right configuration is:

- TIM1_ARR = t1_on + t1_off , TIM1_RCR= 1, TIM1_CCR1= t1_on;

For the second portion of the signal waveform, the right configuration is:

- TIM1_ARR = t2_on + t2_off , TIM1_RCR= 0, TIM1_CCR1= t2_on;

For the third portion of the signal waveform, the right configuration is:

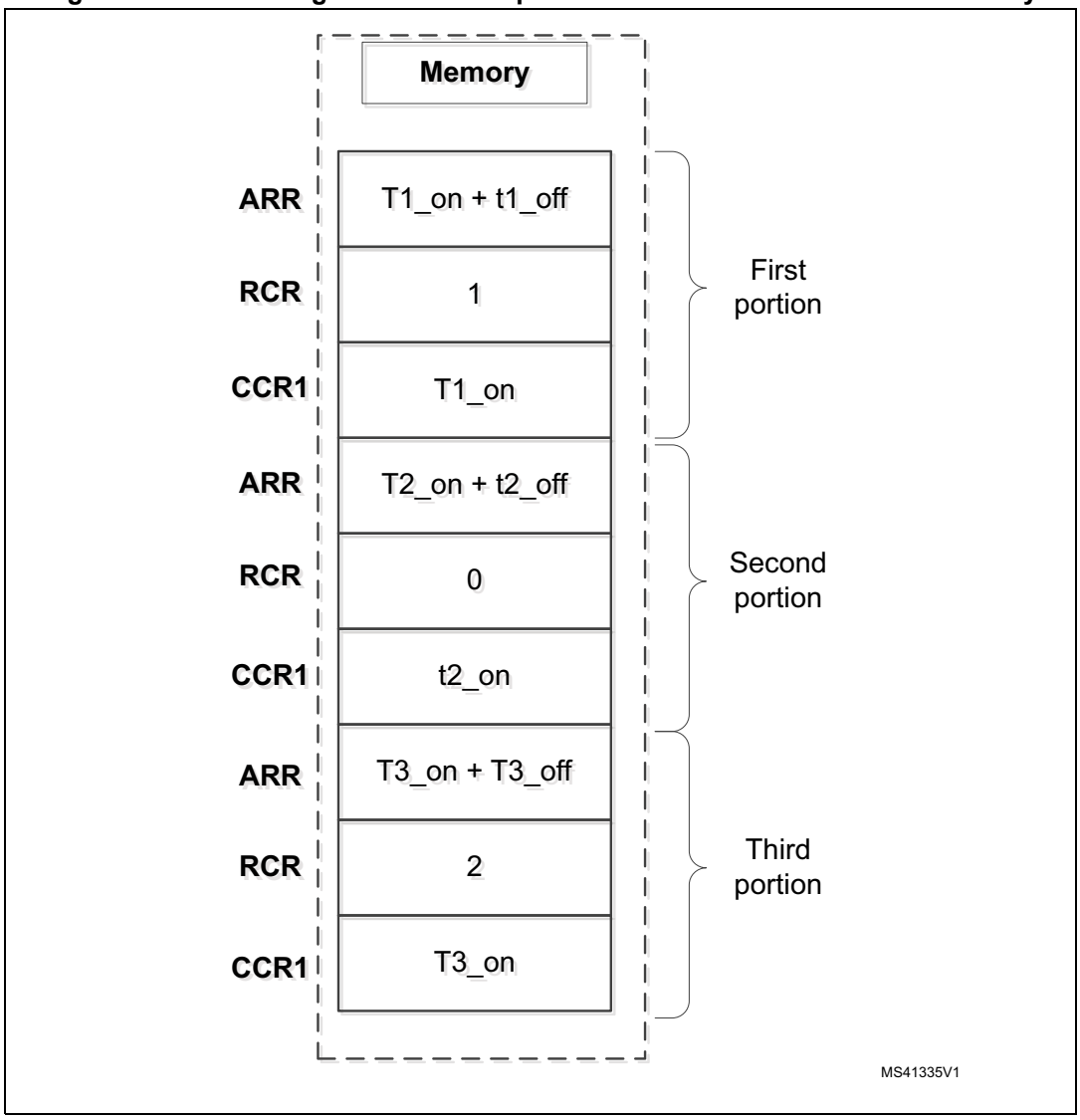
- TIM1_ARR = t3_on + t3_off , TIM1_RCR= 2, TIM1_CCR1= t3_on;

Memory-buffer content:

Considering the above values for TIM1_ARR, TIM1_RCR and TIM1_CCR1 timer registers for each portion of the targeted waveform, refer to *Figure 31* for the values table to be defined in the memory of the microcontroller.

Note that the order of data values within the table is the same as for their respective registers within the registers map of the timer peripheral. The correct order should be TIM1_ARR, TIM1_RCR and then TIM1_CCR1 register.

Figure 31. Waveform generation data pattern stored in microcontroller memory



The C language source code for the above described values table is stated below. The *Const* keyword is used to indicate that the table content is not modified on the fly during the waveform generation.

The memory array is allocated in the Flash memory. If it is required to change the waveform parameters during run-time, the keyword *Const* should not be used. The memory array is allocated in two memory regions: the Flash memory region and the SRAM memory region.

```
uint32_t const aSRC_Buffer[9] = { t1_on+t1_off,1,t1_on,
t2_on+t2_off,0,t2_on, t3_on+t3_off,2,t3_on};
```

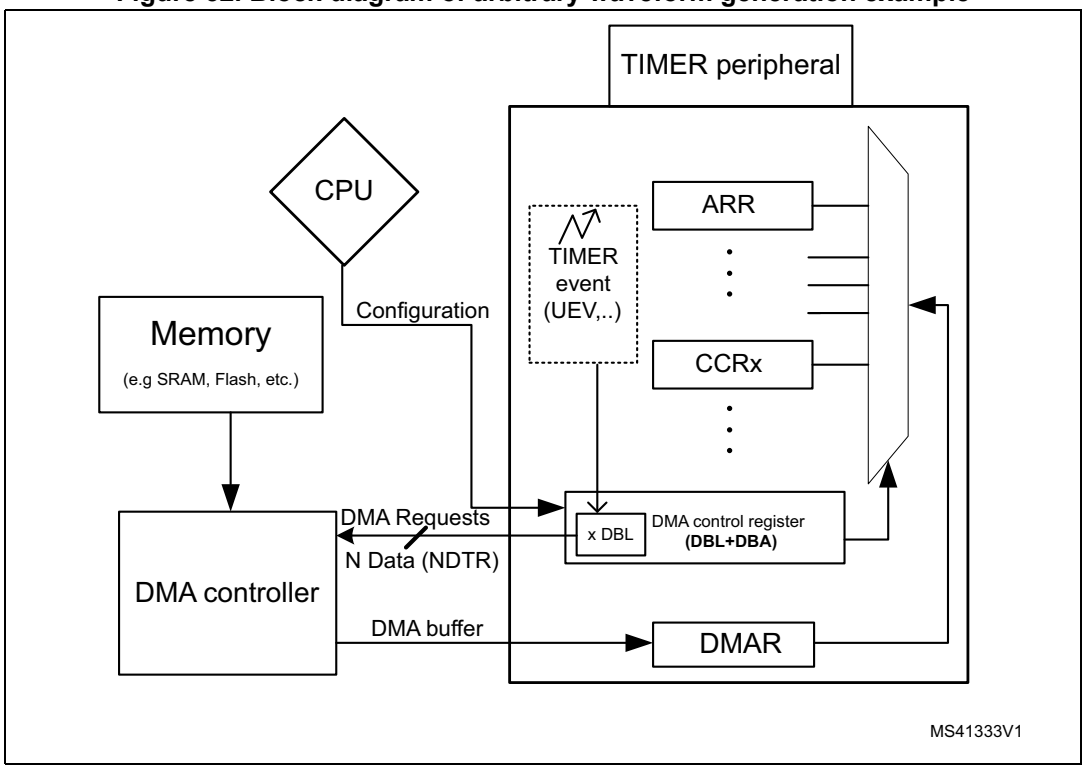
Timer DMA-burst configuration

In order to output the desired waveform, the TIM1 timer DMA-burst should be configured as below:

- As there are three timer registers to update, the burst transfer length is **3.** The DBL[4:0] control bit-field within the TIM1_DCR should be set to 2. Three data transferred each UEV, three update event occur.
- Among the timer registers to update, the timer TIM1_ARR register is the first in the TIM1 timer register map, so it is defined as the base for the transfer. The DBA[4:0] control bit-filed should be set in order to point to the TIM1_ARR register (DBA[4:0] = 11).

Figure 32 illustrates the block diagram for the previous described configurations.

Figure 32. Block diagram of arbitrary waveform generation example



Clock configuration

In this example to get TIM1 counter clock at 32 MHz:

- TIM1 input clock

The TIM1 input clock (TIM1CLK) is set to APB2 clock (PCLK2):

TIM1CLK = PCLK2 and PCLK2 = HCLK => TIM1CLK = HCLK = SystemCoreClock.

- TIM1 Prescaler

To get TIM1 counter clock at 32 MHz, the prescaler is computed as follows:

Prescaler = (TIM1CLK / TIM1 counter clock) - 1

Prescaler = (SystemCoreClock / 32 MHz) – 1

Frequencies and duty cycles calculation

Within this example, the data pattern for the generated waveform is defined as below:

```
uint32_t aSRC_Buffer[9] = { 4000,1,800,10000,0,8500,4000,2,200};
```

And based on the below calculations:

– TIM1 frequency calculation (F1,F2):

$$TIM1Frequency(F1) = \frac{TIM1_{counterclock}}{TIM_{ARR}+1} = \frac{32MHz}{4000+1} = 8.0KHz$$

$$TIM1Frequency(F2) = \frac{TIM1_{counterclock}}{TIM_{ARR}} = \frac{32MHz}{10000+1} = 3.2KHz$$

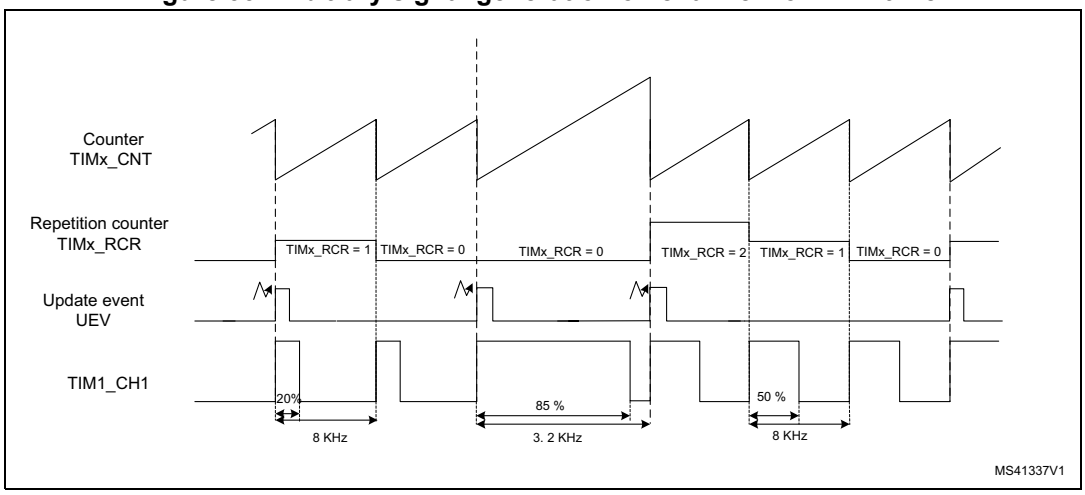
– TIM1 channel1 duty cycles calculation (D1, D2, D3):

$$TIM1dutycycle(D1) = \frac{TIM1_{CCR1}}{TIM_{ARR}} \times 100 = \frac{800}{4000} \times 100 = 20\%$$

$$TIM1dutycycle(D2) = \frac{TIM1_{CCR1}}{TIM_{ARR}} \times 100 = \frac{8500}{10000} \times 100 = 85\%$$

$$TIM1dutycycle(D3) = \frac{TIM1_{CCR1}}{TIM_{ARR}} \times 100 = \frac{2000}{4000} \times 100 = 50\%$$

Figure 33. Arbitrary signal generation on channel1 of TIM1 timer



Firmware description

To achieve the generation of an arbitrary signal described in the previous section, the following steps should be applied:

- **System clock**
  – PLL as system clock source: 64 MHz
  – HSI as oscillator (no need to solder HSE in Nucleo board)
  – AHB div = 1 / APB1 div = 2 / APB2 div= 1

This example uses the DMA1 with TIMER1:

- **DMA2 configuration**

```
/* DMA1 clock enable */
RCC->AHBENR |= RCC_AHBENR_DMA1EN;
/* Configure DMA1 Channel5 CR register */
/* Reset DMA1 Channel5 control register */
  DMA1_Channel5->CCR = 0;
/* Set CHSEL bits according to DMA Channel 5 */
/* Set DIR bits according to Memory to peripheral direction */
/* Set PINC bit according to DMA Peripheral Increment Disable */
/* Set MINC bit according to DMA Memory Increment Enable */
/* Set PSIZE bits according to Peripheral DataSize = Word */
/* Set MSIZE bits according to Memory DataSize Word */
/* Set CIRC bit according to circular mode */
/* Set PL bits according to very high priority */
/* Set MBURST bits according to single memory burst */
/* Set PBURST bits according to single peripheral burst */
  DMA1_Channel5->CCR |= DMA_MEMORY_TO_PERIPH |
                        DMA_PINC_DISABLE| DMA_MINC_ENABLE|
                        DMA_PDATAALIGN_WORD| DMA_MDATAALIGN_WORD |
                        DMA_CIRCULAR | DMA_PRIORITY_HIGH;
/* Write to DMA1 Channel5 number of data's register */
  DMA1_Channel5->CNDTR = 9;
/* Write to DMA1 Channel5 peripheral address register */
  DMA1_Channel5->CPAR = (uint32_t)TIM1_DMAR_ADDRESS;
/* Write to DMA1 Channel5 Memory address register */
/* Set the address to the memory buffer  "aSRC_Buffer" */
  DMA1_Channel5->CMAR = (uint32_t)aSRC_Buffer;
/* Enable DMA1 Channel5 */
  DMA1_Channel5->CCR |= (uint32_t)DMA_CCR_EN;
```

- **TIM1 configuration**

```
/* set the Timer prescaler */
  Tim1Prescaler= (uint16_t) (SystemCoreClock / 32000000) - 1;
/* Configure the period */
  TIM1->ARR = 0xFFFF;
/* Configure the Timer prescaler */
  TIM1->PSC = Tim1Prescaler;
/* Configure pulse width */
  TIM1->CCR1 = 0xFFF;
/* Select the ClockDivison to 1 */
/* Reset clockDivision bit field */
  TIM1->CR1 &= ~ TIM_CR1_CKD;
/* Select DIV1 as clock division*/
```

```
  TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* Select the Up-counting for  TIM1 counter */
/* Reset mode selection bit fiels*/
  TIM1->CR1 &= ~( TIM_CR1_DIR | TIM_CR1_CMS);
/* selct Up-counting mode */
  TIM1->CR1 |=  TIM_COUNTERMODE_UP;
/* SET PWM1 mode */
/* Reset the Output Compare Mode Bits */
  TIM1->CCMR1 &= ~TIM_CCMR1_OC1M;
  TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
/* Select the output compare mode 1*/
  TIM1->CCMR1 |= TIM_OCMODE_PWM1;
/* Enable the output compare 1 Preload */
  TIM1->CCMR1 |= TIM_CCMR1_OC1PE;
/* Enable auto-reload Preload  */
  TIM1->CR1 |= TIM_CR1_ARPE;
/* TIM1 DMA Update enable */
  TIM1->DIER |= TIM_DMA_UPDATE;
/* Configure of the DMA Base register and the DMA Burst Length */
/* Reset DBA and DBL bit fields */
  TIM1->DCR &= ~TIM_DCR_DBA;
  TIM1->DCR &= ~TIM_DCR_DBL;
/* Select the DMA base register and DMA burst length */
  TIM1->DCR = TIM_DMABase_ARR | TIM_DMABurstLength_3Transfers;
/* Enable UEV by setting UG bit to Load buffer data into preload registers
*/
  TIM1->EGR |= TIM_EGR_UG;
/* wait until the RESET of UG bit*/
  while((TIM1->EGR & TIM_EGR_UG) == SET){}
/* Enable UEV by setting UG bit to load data from preload to active
registers */
  TIM1->EGR |= TIM_EGR_UG;
/* Enable the TIM1 Main Output */
  TIM1->BDTR |= TIM_BDTR_MOE;
/* Enable CC1 output*/
  TIM1->CCER |= TIM_CCER_CC1E;
/* Enable the TIM Counter */
  TIM1->CR1 |= TIM_CR1_CEN;
```

- GPIO:
  - Pin PA8: TIM1_ch1_output
  - Mode: push pull
  - Pull: pull-up
  - Speed: high
  - Alternate function: GPIO_AF6_TIM1

6 N-pulse waveform generation using timer synchronization

This application example is split into two parts describing two similar application examples. Both examples use of the inter-timers synchronization to generate an N-pulse waveform but with a small difference:

- Within the first application example, covered by the part 1 of this section, an N-pulse waveform is generated on the output and the complementary output of TIM1 timer channel 1. At the end of the N-pulse waveform generation, the TIM1 timer channel 1 outputs keep their last state where the channel 1 output is low and its complementary output is high.
- Within the second application example, covered by the part 2 of this section, an N-pulse waveform is generated on the output and the complementary output of TIM1 timer channel 1. At the end of the N-pulse waveform generation, the TIM1 timer channel 1 outputs should be both low. This can be achieved by using the commutation feature built-in the STM32 timers. This feature is detailed in the second part of this chapter.

6.1 Timer synchronization overview

Some of the STM32 timers are linked together internally for inter-timer synchronization or chaining. The STM32 timer peripherals featuring the inter-timers synchronization capability have only one synchronization output signal named TRGO (abbreviation for “Trigger Out”). They also have four synchronization inputs (named ITRx where x ranges from 1 to 4) connected to the other STM32 timers synchronization outputs. The reference manual for any STM32 microcontroller lists the inter-timers connection matrix.

Note that only the timers featuring the master/slave controller unit support the inter-timer synchronization (with only few exceptions). For example, the TIM2 timer peripheral features a master/slave controller unit and it can be synchronized with other STM32 timer peripherals.

The TIM2 can trigger events inside other timers through its synchronization TRGO output signal. In this case the TIM2 timer peripheral is acting as a master timer. TIM2 timer peripheral can also be configured to be triggered by other timer synchronization output signals; in this case the TIM2 timer peripheral is acting as a slave timer. A timer peripheral can act as a slave timer and as a master timer simultaneously.

The TIM11 timer peripheral is an example of timer peripheral that does not feature a master/slave controller unit. It does not feature a synchronization of TRGO output signal, but it is able to act as master timer for some other timer peripherals. This is possible through the usage of the TIM11 timer channel output as synchronization output signal. The TIM11 timer channel output is connected to other timer peripherals synchronization inputs.

Timer-master configuration

When a timer peripheral is configured as a master timer, its corresponding synchronization TRGO output signal may output a synchronization pulse next to any of the timer events listed below. Note that the presented list is not exhaustive and only the most common modes are listed. The list of master modes may vary from one microcontroller family to another:

- Reset: the UG bit from the EGR register is used as a trigger output (TRGO)
- Enable: the counter enable signal is used as a trigger output (TRGO). It is used to start several timers at the same time, or to control a window in which a slave timer is enabled
- Update: the update event is selected as trigger output (TRGO). For example, a master timer can be used as a prescaler for a slave timer
- Compare pulse: the trigger output sends a positive pulse when the CC1IF flag is set (even if it was already high) as soon as a capture or a compare match occurs
- OC1Ref: OC1REF signal is used as trigger output (TRGO).
- OC2Ref: OC2REF signal is used as trigger output (TRGO).
- OC3Ref: OC3REF signal is used as trigger output (TRGO).
- OC4Ref: OC4REF signal is used as trigger output (TRGO).
- Encoder clock output: The encoder clock signal is used as a trigger output (TIM_TRGO)[(a)].

To configure the timer event or internal signal to be used as a synchronization output, the right value should be written to the MMS (master mode selection) control bit-field within the TIMx_CR2 timer control register 2.

Timer-slave configuration

Each timer peripheral that embeds a master/slave controller unit features four synchronization inputs already connected to the other timers synchronization outputs. Note that only one synchronization input can be active at each time and the TS (trigger selection) control bit-field is used to select which synchronization input is the active one.

The detection of an active edge on the synchronization input of one timer peripheral may trigger one of timer event inside the peripheral like an “update event” and counter reset, or counter increment. This depends on the configured value into the SMS (slave mode selection) control bit-field within the timer slave mode control register (TIMx_SMCR).

To select which synchronization input to use, the slave timer is connected to the master timer through the input trigger. Each ITRx is connected internally to another timer, and this connection is specific for each STM32 product.

STM32 timer-commutation feature

The commutation feature is used in combination with the preload feature to change the timer channel-configuration in perfect synchronization with timer external events that are fed into it through one of its internal or external inputs. The configuration to which this refer, can be for example channel output mode, channel enabled/disabled or other. ITRx inputs are an example of internal inputs and ETR, TI1, or TI2 are examples of external inputs.

a. Only available in the STM32C5 and STM32G5 series.

As stated in *Section 1: Basic operating modes of STM32 general-purpose timers*, the OCxM, CCxE and CCxNE control bit-fields feature the preload capability. When the preload feature on these bit-fields is enabled, any write access to them does not change the timer channel operating mode. The reason is that the write operation is performed on the preload instances of these control bit-fields. Their active instances, which effectively control the timer channel output mode, remain unchanged.

As soon as a commutation event is generated inside the timer, the content of the preload instances of these control bit-fields is transferred into the active instances and consequentially the channel output mode may be changed.

Depending on the configuration of the CCUS control bit-field within the timer control register 2 (TIMx_CR2), the commutation event can be generated after detecting an active edge on the timer trigger input signal (TRGI). In particular, it can be generated after detecting an active edge on the timer active synchronization ITRx input of the timer peripheral (ITRx inputs are part of the sources of TRGI signal).

This capability may be used to make one timer control when a commutation event on a second timer should be triggered through inter-timers synchronization. Alternatively, the commutation event may be generated by software through the setting of the COMG control bit-field within the TIMx_EGR timer register.

6.2 N-pulse waveform generation application example - part 1

This example demonstrates how to use the STM32 timer interconnection feature in order to generate a specific number of pulses in each configurable period.

Application overview

The application is designed around the Nucleo board NUCLEO-F302R8, based on STM32F302x8 microcontroller, as the board main controller. Nevertheless, this application can be ported easily to any STM32 hardware platform. The Nucleo board embeds its own ST-Link/V2 debugger. Only a USB cable is needed to interface the Nucleo board with a PC computer for application-binary-image download into the microcontroller and for application debug purposes.

The *Figure 34* shows the synoptic diagram for the periodic N-pulses generation application presented by this application example. The generated signal is outputted on the PA.08 IO of the STM32F302 microcontroller, which is mapped on the pin 8 of the CN9 connector.

The main features described are:

- TIMER 2 is configured as master trigger mode to trig TIMER1
- TIMER 1 in configured as slave one pulse (OPM) mode.

Figure 34. Periodic N-pulses generation block diagram

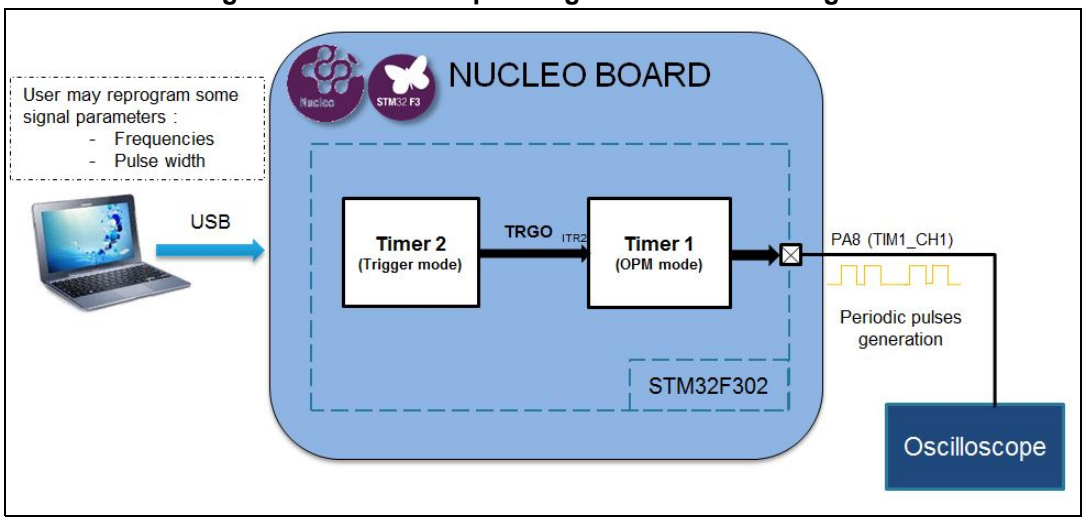
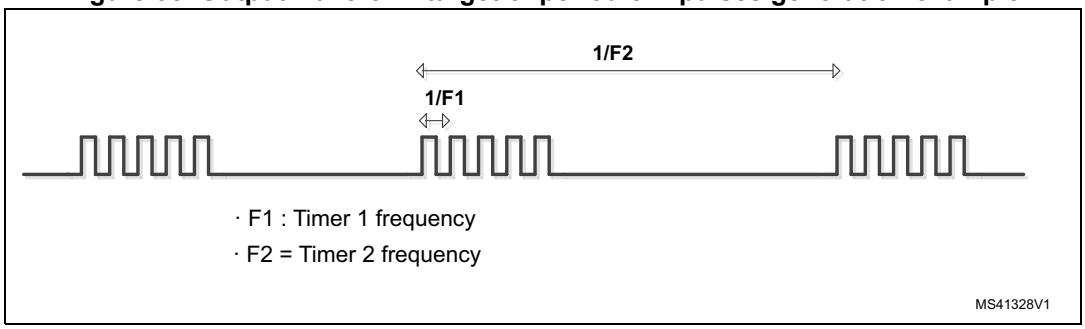


Figure 35 shows the targeted waveform that we want to generate in this example: generating N=5 pulses periodically.

Figure 35. Output waveform target of periodic N-pulses generation example



The waveform described by this example is designed to output six pulses periodically on each PWM cycle, similar to waveform signal on *Figure 35*. The waveform is made by two frequencies given by two programmed periods: T2 is TIM2 timer peripheral period and T1 is TIM 1 timer peripheral period. The source code of this application can be easily reshaped to output different waveforms, but for demonstration purposes the one illustrated on *Figure 37* is targeted.

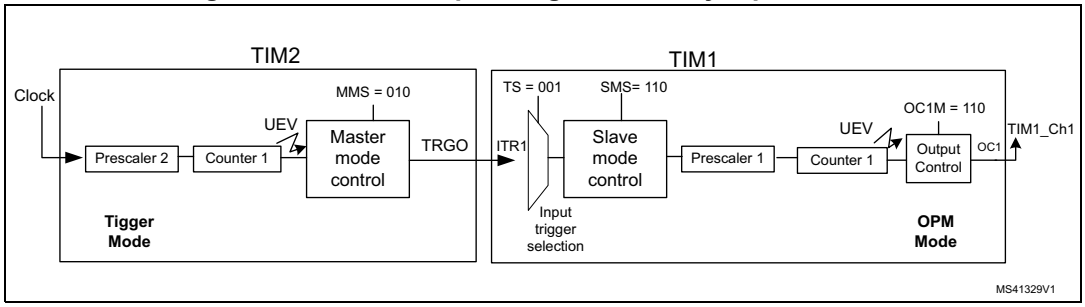
Functional description

In reference to *Figure 35*, the targeted waveform is a PWM signal made by a periodic N-pulses for (six pulses in this example). The pulses are made with a certain on-time and off-time: t1_on and t1_off, t2_on and the sum is equal to T1. Each T2 period, the pulses are generated again, and the rest of the PWM cycle maintains a low level.

To make an STM32 timer peripheral to output these periodic pulses, it is done by using two timer peripherals interconnected: TIM1 timer peripheral and TIM2 timer peripheral. TIM1 timer peripheral generates a PWM signal by a T1_on duty cycle programmed in TIM1_CCR1 and T1 period programmed in TIM1_ARR. The TIM2 timer peripheral ensures that the period of each generation cycle is equal to T2, which is programmed in its TIM2_ARR register.

As shown in *Figure 36*, TIM2 peripheral and TIM3 timer peripheral are interconnected as master and slave successively.

Figure 36. Periodic N-pulses generation synoptic schema



The TIM2 peripheral timer is configured as master trigger mode to trig the TIM1 peripheral ITR1 timer input via its internal TRGO signal, at each update event. The time between two “update event” is the T2 of the waveform targeted.

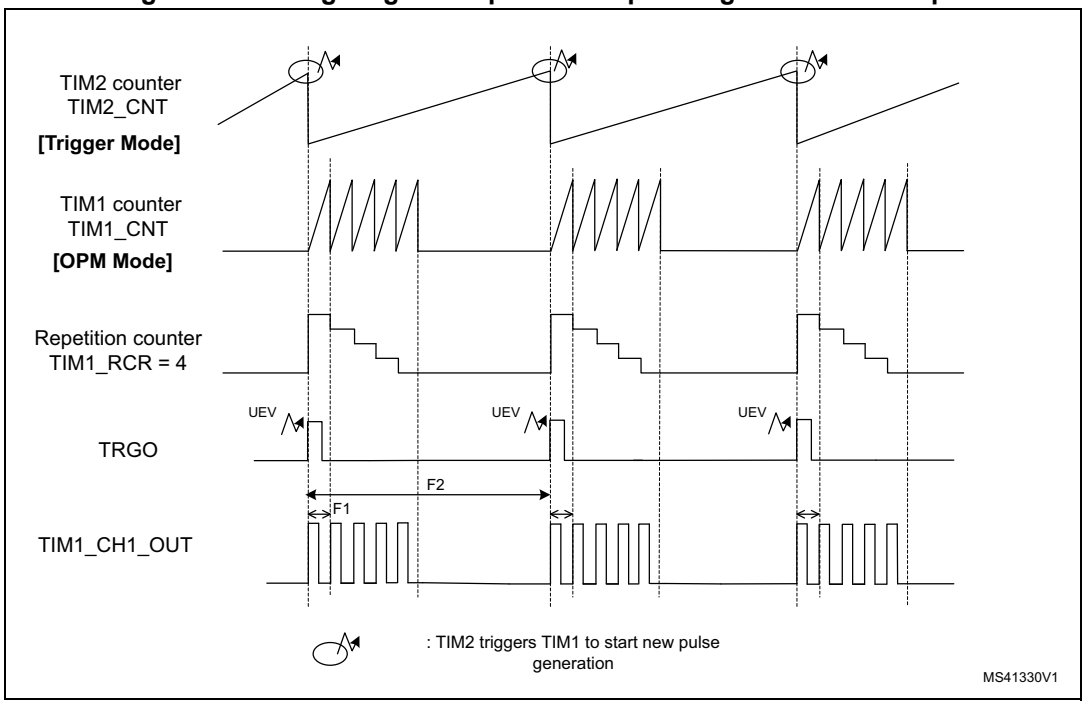
The TIM1 peripheral timer is configured as slave one pulse (OPM) mode, to generate one pulse when triggered by TIM2 timer peripheral.

When a rising edge occurs on the trigger input (ITR1) the counter starts up-counting for only one time until it matches the configure period (TIM1_CCR1) equal to 1/F1 and then an UEV occurs.

To repeat the pulse generated (the up-counting), we use the **repetition counter** of TIM1 timer peripheral which should be configured to the number of pulses per the on-going signal waveform portion minus one (TIM1_RCR = number_of_pulses_per_portion – 1 = 6-1). The UEV is only generated after up-counting is repeated for a number of time configured TIM1_RCR +1.

The timing diagram on *Figure 37* shows how the synchronized timers with the describer configuration allow to get the desired waveform.

Figure 37. Timing diagram of periodic N-pulses generation example



Firmware description

- **System clock**
  - PLL as System clock source: 64 MHz
  - HSI as oscillator (no need to solder HSE in Nucleo boards)
  - AHB div = 1 / APB1 div = 2 / APB2 div= 1
- **TIMER 1**

  We use TIM1_channel1 on output compare mode to generate PWM1 signal.

  - APB2 prescaler = 64 -1 (to get 1 MHz as timer clock)
  - Period (ARR) = 50 µs -> frequency F1 = 20 kHz
  - Pulse (CCR1) = period / 2 (50%)
  - RCR = 5-1 -> to get 5 pulses repeated
  - One pulse mode selected
  - Slave mode trigger selected
  - Input trigger: ITR1
  - PWM mode: mode1
  - Counting mode: up-counting
  - Output compare preload: enabled

```
/* set the Timer prescaler to get 1MHz as counter clock */
  Tim1Prescaler= (uint16_t) (SystemCoreClock / 1000000) - 1;
/* Initialize the PWM period to get 20 kHz as frequency from 1MHz */
  Period = 1000000 / 20000;
/* configure the Timer prescaler */
```

```
  TIM1->PSC = Tim1Prescaler;
/* configure the period */
  TIM1->ARR = Period-1;
/* configure the repetition counter */
  TIM1->RCR = ((uint32_t) 6) -1;
/* configure pulse width */
  TIM1->CCR1 = Period / 2;
/* Select the Clock Division to 1*/
/* Reset clock Division bit field */
  TIM1->CR1 &= ~ TIM_CR1_CKD;
/* Select DIV1 as clock division*/
  TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* Select the Up-counting for TIM1 counter */
/* Reset mode selection bit fields*/
   TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* select Up-counting mode */
  TIM1->CR1 |= TIM_COUNTERMODE_UP;
/* SET PWM1 mode */
/* Reset the Output Compare Mode Bits */
  TIM1->CCMR1 &= ~TIM_CCMR1_OC1M;
  TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
/* Select the output compare mode 1*/
  TIM1->CCMR1 |= TIM_OCMODE_PWM1;
/************** One pulse mode configuration ************/
  /* One Pulse Mode selection */
  TIM1->CR1 |= TIM_CR1_OPM;
/*****************************************************/
/******* Slave mode configuration: Trigger mode *********/
/* Select the TIM_TS_ITR1 signal as Input trigger for the TIM */
  TIM1->SMCR &= ~TIM_SMCR_TS;
  TIM1->SMCR |= TIM_TS_ITR1;
/* Select the Slave Mode */
  TIM1->SMCR &= ~TIM_SMCR_SMS;
  TIM1->SMCR |= TIM_SLAVEMODE_TRIGGER;
/*****************************************************************/
/* Enable the output compare 1 Preload */
  TIM1->CCMR1 |= TIM_CCMR1_OC1PE;
/* Set the UG bit to enable UEV */
  TIM1->EGR |= TIM_EGR_UG;
/* Enable the TIM1 Main Output */
  TIM1->BDTR |= TIM_BDTR_MOE;
/* Select active low as output polarity level */
/* Reset the Output Polarity level */
  TIM1->CCER &= ~TIM_CCER_CC1P;
/* Set the Low output */
```

```
  TIM1->CCER |= TIM_OCPOLARITY_LOW;
/* Enable CC1 output on High level */
  TIM1->CCER |= TIM_CCER_CC1E;
/* Enable the TIM Counter */
  TIM1->CR1 |= TIM_CR1_CEN;
```

- **TIMER 2**

We use TIM2_channel1 on output compare mode to generate PWM1 signal.

  - APB1 prescaler = 64 -1 (to get 1 MHz as timer clock)
  - Period (ARR) = 20 000 us -> frequency F2 = 50 Hz
  - Pulse (CCR1) = period /2 (50%)
  - Master mode selected: TRGO update
  - Counting mode: up-counting

```
/* set the Timer prescaler to get 1MHz as counter clock */
  Tim2Prescaler= (uint16_t) ((SystemCoreClock ) / 1000000) - 1;
/* Initialize the PWM period to get 50 Hz as frequency from 1MHz */
  Period = 1000000 / 50;
/* configure the period */
  TIM2->ARR = Period-1;
/* configure the Timer prescaler */
  TIM2->PSC = Tim2Prescaler;
/* Select the Clock Divison to 1*/
/* Reset clock Division bit field */
  TIM2->CR1 &= ~ TIM_CR1_CKD;
/* Select DIV1 as clock division*/
  TIM2->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* Select the Up-counting for  TIM1 counter */
/* Reset mode selection bit fields */
  TIM2->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* select Up-counting mode */
  TIM2->CR1 |= TIM_COUNTERMODE_UP;
/****** Master mode configuration: Trigger update mode *******/
/* Trigger of TIM2 Update into TIM1 Slave */
  TIM1->CR2 &= ~ TIM_CR2_MMS;
  TIM2->CR2 |= TIM_TRGO_UPDATE;
/*****************************************************************/
/* Enable the TIM Counter */
  TIM2->CR1 |= TIM_CR1_CEN;
```

- GPIO
  - Pin PA8: TIM1_ch1_output
  - Mode: push pull
  - Pull: pull-up
  - Speed: high
  - Alternate function: GPIO_AF6_TIM1

6.3 N-pulse waveform generation application example - part2

Application overview

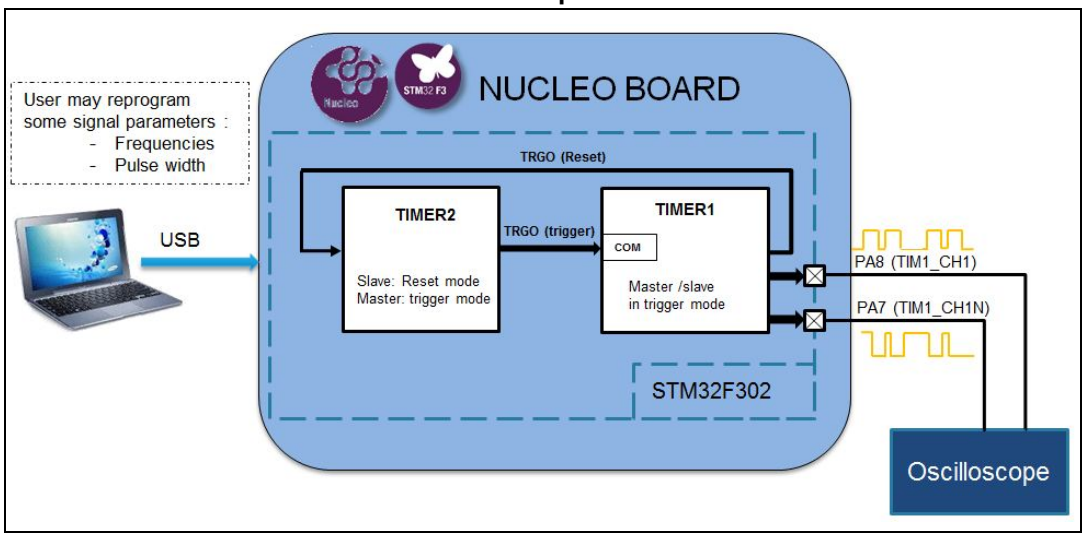
This application example is designed around the Nucleo board NUCLEO-F302R8, based on STM32F302x8 microcontroller, as the board main controller. Nevertheless, this application can be ported easily to any STM32 hardware platform. The Nucleo board embeds its own ST-Link/V2 debugger. Only a USB cable is needed to interface the Nucleo board with a PC computer for application-binary-image download into the microcontroller and for application debug purposes.

Figure 38 shows the synoptic diagram for the periodic N-pulses generation application presented by this application example. Both complementary generated signals are outputted on the PA.08 IO and PA.07 IO of the STM32F302 microcontroller, which are mapped respectively on the pin 8 of the CN9 connector and on the pin 26 of the CN10 connector.

The main features described are:

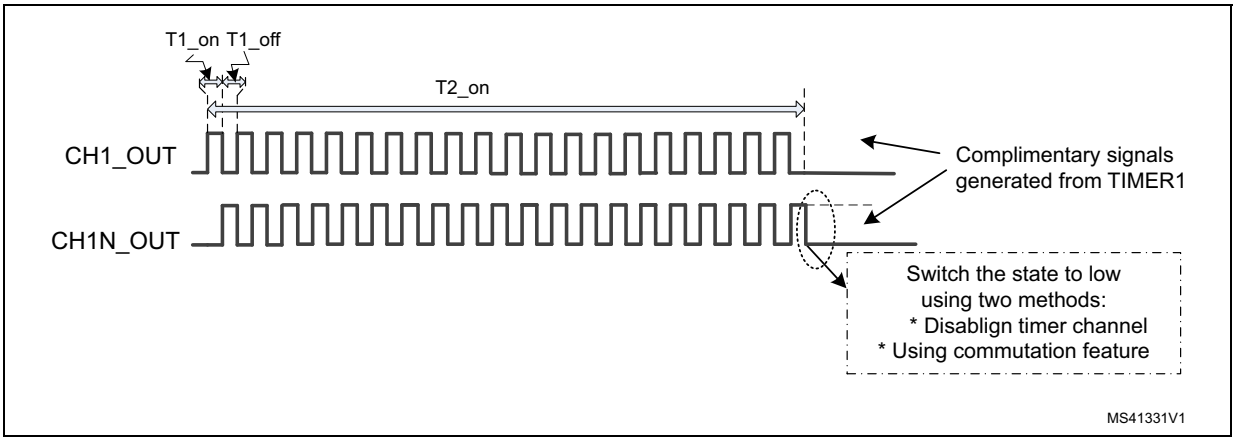
- TIMER 2 is configured as slave-reset mode and as master-trigger mode
- TIMER 1 is configured as slave and master in trigger mode

Figure 38. Synoptic schema of two complementary N pulses waveform generation example



The two complementary N-pulses waveforms generator described by this example is designed to output two complimentary waveforms made by N-pulses each one and which have the same final state. The result should be similar to the signals shown on *Figure 39*. The source code for this application can be easily reshaped to output different waveforms, but for demonstration purposes, the target is the one illustrated on *Figure 39*.

Figure 39. Output of two N-pulses complimentary waveforms generation example



Functional description

As per the waveforms shown in *Figure 39*, the targeted waveforms are two complimentary PWM signals made by only N-pulses for 20 pulses in our case. The pulses are made with a certain on-time and off-time, t1_on and t1_off. T2_on is the active time of pulses frame generation. Both complimentary signals have to end by the same final state, this is low level in our example as shown on *Figure 39*.

To make an STM32 timer peripheral output these PWMs, two interconnected timer peripherals should be used. One of them work as complimentary PWMs generator and the other one controls the end of the pulses generation.

A first timer control the start and stop timing of the pulses generation by counting T2_on and by triggering the generator timer in order to shut down the PWMs output. Therefore, both timer peripheral counters start counting at the same time by synchronizing the start timing.

The intuitive way to do this is by configuring the generator timer as master trigger mode and the other one as slave reset mode, and to reset the first timer peripheral counter when the generator timer is enabled. In the other hand, the generated timer is configured also as a slave-trigger mode and the first timer peripheral as master-trigger update mode to generate an “update event” when T2_on time counting overflows and stops PWM generation.

When T2_on counting ends and an “update event” is generated to trigger the generator timer, there are two ways to stop the PWMs generation in the interrupt routine: either disabling both capture/compare outputs, or by forcing the outputs to inactive mode using the commutation feature.

The commutation update control in TIM1 timer peripheral should be enable and configured to trigger its TRGI input by setting CCUS bit-field in the TIM1_CR2 peripheral. The timer commutation interrupt source is enabled by setting the COMIE bit in the TIM1_DIER register. The configuration for the next step of COM event should be programmed in advance, this configuration is forced to inactive level by setting the OC1M [2:0] bit-field of TIM1_CCMR1 register to 4.

When TIM2 triggers a TRGI TIM1 input, a commutation event (COM) occurs by detecting a rising edge. Then, the preload OC1M bit-field is transferred to the shadow bits at the COM commutation event. Finally, the TIM1 outputs are forced low simultaneously.

Both timers should be configurable as master/slave mode and the generator timer features complimentary outputs are needed as well. For this example, the TIM1 timer peripheral is a good candidate for the timer generator role as it features four timer channels that can output

both the complimentary signals and the commutation event. The TIM1 timer channel1 and channel1N are used. We use TIM2 for the other timer.

In order to prevent the timer channel from generating unwanted glitches due to updating the timer channel register, the capture compare preload feature of the STM32 timer peripherals is used.

To avoid wasted time cycles before TIM1 timer peripheral, an “update event” is generated after enabling its channel register, the update generation bit-field (UG) in TIM1_EGR register should be set. This makes synchronization timing between both timers peripherals counters more accurate.

To output the waveform shown on *Figure 39* on the TIM1 timer peripheral complimentary channels, both TIM1 timer peripheral and TIM2 timer peripheral are configured as described below:

TIM1 timer peripheral: master trigger-reset mode / slave-trigger mode

The TIMER 1 is configured as master trigger-reset mode to trigger TIM2 timer peripheral and reset its counter. This is done in order to synchronize the counting start time of both timer peripherals counter. The TIMER 1 is configured as slave-trigger mode to be triggered by an update event when TIM2 finish counting T2_on period in order to get down the signal to a low level.

- **TIM1_ARR:** contains the period of the on-going pulse; sum of t1_on and t2_off periods
- **TIM1_CCR1**: contains the duration of the t1_on period of pulse
- **TIM1 timer peripheral**: master trigger-update / slave-trigger reset mode

The timer 2 is configured as master trigger mode to trigger the TIMER1 by an “update event” when T2_on period ends, through the TRGO signal. Timer 2 is configured as slave-trigger reset mode to be reset by TIM1 when enabled.

- **TIM2_ARR:** contains the T2_on period.

6.3.1 Clock configuration

This example aims to get TIM1 and TIM2 counters clock at 1 MHz.

- **TIM1 input clock**

The TIM1 input clock (TIM1CLK) is set to APB2 clock (PCLK2):

TIM1CLK = PCLK2 and PCLK2 = HCLK => TIM1CLK = HCLK = SystemCoreClock = 64 MHz

- **TIM2 input clock**

The TIM2 input clock (TIM2CLK) is set to APB2 clock (PCLK2):

TIM2CLK = PCLK2 and PCLK2 = HCLK => TIM2CLK = HCLK = SystemCoreClock = 64 MHz

- **TIM1 prescaler**

To get TIM1 counter clock at 1 MHz, the prescaler is computed as follows:

Prescaler = (TIM1CLK / TIM1 counter clock) - 1

Prescaler = (SystemCoreClock /1 MHz) – 1

- **TIM2 prescaler**

To get TIM2 counter clock at 1 MHz, the prescaler is computed as follows:

Prescaler = (TIM2CLK / TIM2 counter clock) - 1

Prescaler = (SystemCoreClock /1 MHz) – 1

In this example, the objective is that TIM1 timer peripheral outputs 20 pulses, each pulse with a period (t1_on +t1_of) of 500 us and duty cycle (t1_on) of 50%. Therefore, the timer peripheral should be programmed as below:

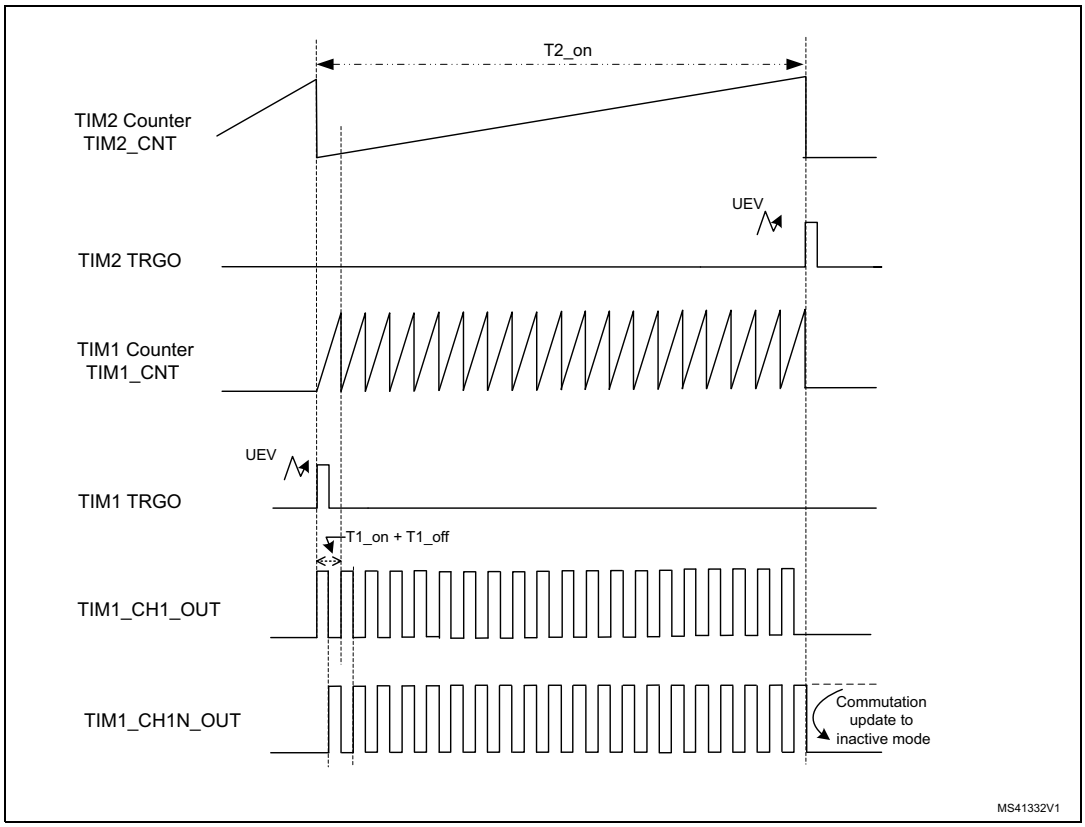
- **TIM1 period**
  - TIM1_ARR = T1_on+T1_off = 500 µs
- **TIM1 duty cycle**
  - TIM1_CCR1 = 250 µs

The calculation of the period that we program on TIM2 timer peripheral which allow as to obtain accurately the 20 pulses is shown below:

- **TIM2 period**
  - TIM2_ARR=T2_ON = 500 x 20 = 10000 µs

The timing diagram on *Figure 40* shows how the synchronized TIMER with the described configuration allows the user to get the desired waveform.

Figure 40. Timing diagram of complimentary N-pulses waveforms generation with similar final state



Firmware description

- System clock
  - PLL as system clock source: 64 MHz
  - HSI as oscillator (no need to solder HSE in Nucleo boards)
  - AHB div = 1 / APB1 div = 2 / APB2 div = 1
- **TIMER 1**

TIM1_channel1 is used on output-compare mode to generate PWM1 signal.

  - APB2 prescaler = 64 -1 (to get 1 MHz as timer clock)
  - Period (ARR) = 50
  - 0 us -> frequency F1 = 2 kHz
  - Pulse (CCR1) = period / 2 (50%)
  - PWM mode: mode1
  - Output compare preload: enabled
  - Set the update generation event
  - Enable the commutation feature
  - Counting mode: up-counting
  - Master trigger reset mode selected
  - Input trigger: ITR1
  - Slave-mode trigger selected

```
/* set the Timer prescaler to get 1MHz as counter clock, SystemCoreClock =
64 MHz */
  Tim1Prescaler= (uint16_t) (SystemCoreClock / 1000000) - 1;
/* Initialize the PWM periode to get 2 kHz as frequency 10000*/
Period = 1000000 / 2000;
/* configure the Timer prescaler */
  TIM1->PSC = Tim1Prescaler;
/* configure the period */
  TIM1->ARR = Period-1;
/* configure pulse width */
  TIM1->CCR1 = Period / 2;
/* Select the Clock Divison to 1*/
/* Reset clock Division bit field */
  TIM1->CR1 &= ~ TIM_CR1_CKD;
/* Select DIV1 as clock division*/
  TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* Select the Up-counting for TIM1 counter */
/* Reset mode selection bit fields */
  TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* select Up-counting mode */
  TIM1->CR1 |= TIM_COUNTERMODE_UP;
/* SET PWM1 mode */
/* Reset the Output Compare Mode Bits */
  TIM1->CCMR1 &= ~TIM_CCMR1_OC1M;
```

```
  TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
/* Select the output compare mode 1*/
  TIM1->CCMR1 |= TIM_OCMODE_PWM1;
/* Select active High as output polarity level */
/* Reset the Output Polarity level */
  TIM1->CCER &= ~TIM_CCER_CC1P;
/* Set the high Output Compare Polarity */
  TIM1->CCER |= TIM_OCPOLARITY_HIGH;
/* Enable CC1 output on High level*/
  TIM1->CCER |= TIM_CCER_CC1E;
/* Select active High as output Complementary polarity level */
/* Reset the Output N State */
  TIM1->CCER &= ~TIM_CCER_CC1NP;
/* Set the Output N Polarity to high level */
  TIM1->CCER |= TIM_OCNPOLARITY_HIGH;
/* Enable CC1 output on High level*/
  TIM1->CCER |= TIM_CCER_CC1NE;
/******************* COM Control update configuration ******************/
/* Set the capture Compare Preload */
 TIM1->CR2 |= TIM_CR2_CCPC;
/* Set CCUS bit to select the COM control update to trigger input TRGI*/
  TIM1->CR2 |= TIM_CR2_CCUS;
/* Enable the Commutation Interrupt sources */
  TIM1->DIER |= TIM_IT_COM;
/********** Master mode configuration: Trigger Reset mode *********/
/* configure TIM1's trigger output Update to trig TIM2 */
  TIM1->CR2 &= ~ TIM_CR2_MMS;
  TIM1->CR2 |= TIM_TRGO_RESET;
/************ Slave mode configuration: Trigger mode ************/
/* Select the TIM_TS_ITR1 signal as Input trigger for the TIM */
  TIM1->SMCR &= ~TIM_SMCR_TS;
  TIM1->SMCR |= TIM_TS_ITR1;
/* Select the Slave Mode */
  TIM1->SMCR &= ~TIM_SMCR_SMS;
  TIM1->SMCR |= TIM_SLAVEMODE_TRIGGER;
/*****************************************************************/
/* Set the UG bit to enable UEV */
  TIM1->EGR |= TIM_EGR_UG;
/* Enable the TIM1 Main Output */
  TIM1->BDTR |= TIM_BDTR_MOE;
/* Enable the TIM Counter */
  TIM1->CR1 |= TIM_CR1_CEN;
```

- **TIMER 2**

TIM2_channel1 is used on output-compare mode to generate PWM1 signal.

- APB1 prescaler = 64 -1 (to get 1 MHz as timer clock)
- Period (ARR) = 10 000 µs
- Counting mode: up-counting
- Master trigger update mode selected: TRGO update
- Input trigger: ITR0
- Slave trigger reset mode selected

```
/* set the Timer prescaler to get 1MHz as counter clock; SystemCoreClock =
64 MHz */
  Tim2Prescaler= (uint16_t) ((SystemCoreClock) / 1000000) - 1;
/* Initialize the PWM periode to get 100 Hz as frequency from 1MHz */
  Period = 1000000 / 100;
/* configure the period */
  TIM2->ARR = Period-1;
/* configure the Timer prescaler */
  TIM2->PSC = Tim2Prescaler;
/* Select the Clock Divison to 1 */
/* Reset clock Division bit field */
  TIM2->CR1 &= ~ TIM_CR1_CKD;
/* Select DIV1 as clock division*/
  TIM2->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* Select the Up-counting for TIM1 counter */
/* Reset mode selection bit fields */
  TIM2->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* select Up-counting mode */
  TIM2->CR1 |= TIM_COUNTERMODE_UP;
/********** Master mode configuration: trigger update ***********/
/* Trigger of TIM2 Update into TIM1 Slave */
  TIM2->CR2 &= ~TIM_CR2_MMS;
  TIM2->CR2 |= TIM_TRGO_UPDATE;
/********** Slave mode configuration: Trigger mode ************/
/* Select the TIM_TS_ITR0 signal as Input trigger for the TIM */
  TIM2->SMCR &= ~TIM_SMCR_TS;
  TIM2->SMCR |= TIM_TS_ITR0;
/* Slave Mode selection: Trigger reset Mode */
  TIM2->SMCR &= ~TIM_SMCR_SMS;
  TIM2->SMCR |= TIM_SLAVEMODE_RESET;
/*****************************************************************/
/* Enable the TIM1 Counter */
  TIM2->CR1 |= TIM_CR1_CEN;
```

- **GPIO**
  - Pin PA8: TIM1_CH1 output
  - Pin PA7: TIM1_CH1N output
  - Mode: push-pull
  - Pull: pull-up
  - Speed: high
  - Alternate function: GPIO_AF6_TIM1
- **Commutation update control**

```
/* Reset the OC1M bits in the CCMR1 register */
TIM1->CCMR1 &= TIM_CCMR1_OC2M;
/* configure the OC1M bits in the CCMRx register to inactive mode*/
TIM1->CCMR1 |= TIM_OCMODE_FORCED_INACTIVE;
```

7 Revision history

Table 1. Document revision history

Date	Revision	Changes
23-Jun-2016	1	Initial release.
03-May-2017	2	Updated *Section 1.4.2: The preload feature of the timer registers* and *Figure 6: Preload mechanism for timer channel register - enabled*.
11-Jul-2019	3	Updated document scope to all STM32 devices and deleted Table 1: Applicable products on cover page.
23-Feb-2026	4	Updated the entire document to cover STM32C5 series.
16-Oct-2026	5	Updated the entire document to cover STM32G5 series.

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

The purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of the purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2026 STMicroelectronics – All rights reserved