

# Kinetis Flextimer Buffered Registers Update

By: Rastislav Pavlanin

## 1. Introduction

The FlexTimer (FTM) module that is widely used in the Kinetis portfolio is a simple timer (commonly known as TPM—mostly used by the Kinetis L and HCS08 devices) with the FLEX functionality extension. The FLEX functionality is added to meet the more complex motor-control as well as power-conversion application demands.

One of the extended features is the possibility to update the buffered registers (MOD, CNTIN, CnV, OUTMASK, INVCTRL, SWOCTRL, etc.) by a specified synchronization event. This application note describes the update of specific registers with their buffered values using the software/hardware trigger synchronization event. This document provides guidance on configuring the FTM module to update the required registers using PWM synchronization, and demonstrates it using figures.

The code examples do not demonstrate real motor-control or power-converter application use cases. They only provide the forms of FTM configuration to demonstrate the features as simply as possible.

## Contents

1.	Introduction .....	1
2.	Buffered Registers in FTM Module .....	2
3.	FTM has no Clock Source (CLKS = 0) .....	2
4.	TPM Approach (FTMEN = 0) .....	2
4.1.	CNTIN register update (FTMEN = 0 or CNTINC = 0) .....	2
4.2.	MOD register update (FTMEN = 0) .....	3
4.3.	CnV register update (FTMEN = 0) .....	3
5.	FTM Approach (FTMEN = 1) .....	4
5.1.	Legacy PWM synchronization (SYNCMODE = 0) .....	5
5.2.	Enhanced PWM synchronization (SYNCMODE = 1) .....	7
6.	Read Coherency Mechanism used in Dual Edge Capture Mode (FTMEN = 1, DECAPEN = 1) .....	23
7.	Demonstration Using Sample Code Example .....	25
7.1.	Peripherals configuration .....	25
7.2.	Running the example .....	27
7.3.	Code execution flow .....	28
7.4.	Selected examples from sample code .....	29
8.	References .....	36
9.	Revision History .....	36



## 2. Buffered Registers in FTM Module

The FTM modules available on Kinetis devices contain specific buffered registers. Only the MOD, CNTIN, CnV, OUTMASK, INVCTRL, and SWOCTRL registers support this feature. When the core (or DMA) tries to write into the register, the value is first written into the buffer. The update of a register by its buffer value is realized by three different approaches that are dependent on the clock source, the FTM operation mode (input capture, output compare, PWM mode, or dual edge capture mode), and the FTM configuration (FTMEN, SYNCMODE, etc.).

### NOTE

In all other cases (than the one mentioned above), the registers are updated by their buffered values in the next system clock cycle after writing into the register.

## 3. FTM has no Clock Source (CLKS = 0)

In this case, the FTM counter is not counting. Writing to the buffered register takes place immediately after writing to the register. The buffer has no purpose here. This is not dependent on the FTM feature activation (FTMEN = X).

## 4. TPM Approach (FTMEN = 0)

The FTM module acts as a simple TPM module when the FlexTimer feature is disabled (FTMEN in the FTMx\_MODE register is 0). In such case, the registers that represent the FlexTimer features (such as OUTMASK, INVCTRL, SWOCTRL, etc.) are not in use. The only three buffered registers that you can use are CNTIN, MOD, and CnV.

### 4.1. CNTIN register update (FTMEN = 0 or CNTINC = 0)

The CNTIN register is updated by its buffer value in the next system clock cycle after writing to the register (see the following figure). The same behavior occurs when the CNTIN register synchronization bit in the FTMx\_SYNCONF register is disabled (CNTINC = 0), this is not dependent on the FTMEN bit.

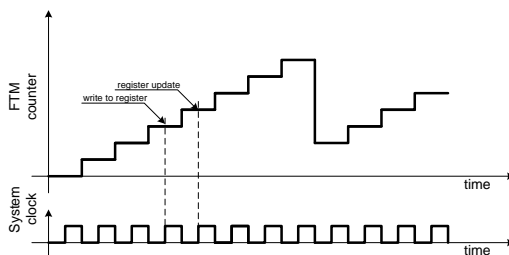


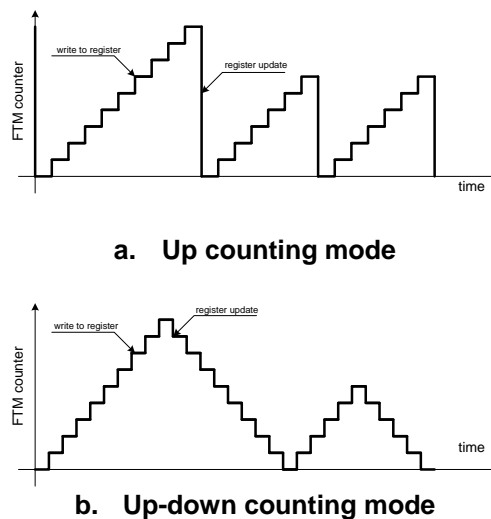
Figure 1. CNTIN register update when FTMEN = 0 or CNTINC = 0

**NOTE**

The FTM counter clock does not have to be the same as the system clock.  
 This depends on the clock source selection as well as the pre-scaler factor.  
 The FTM counter clock must not exceed the system clock.

**4.2. MOD register update (FTMEN = 0)**

The MOD register is updated by its buffer value after the register is written to and the counter changes from the MOD value (not in the buffer but in the register) to the next counter value (see the following figure). The next counter value depends on the counting mode (up counting or up-down counting).



**Figure 2. MOD register update when FTMEN = 0**

**4.3. CnV register update (FTMEN = 0)**

The CnV register update depends on the FTM mode.

When the FTM is configured to work in the output compare mode, then the CnV register is updated on the next counter change after writing to the register (see [Figure 3 a.](#)).

When the FTM is configured to work in the PWM mode, then the register is updated by its buffer value after writing to the register, and the counter changes from the MOD value (not in the buffer but in the register) to the next counter value (see [Figure 3 b.](#) and [Figure 3 c.](#)). The next counter value depends on the counting mode (up counting or up-down counting).

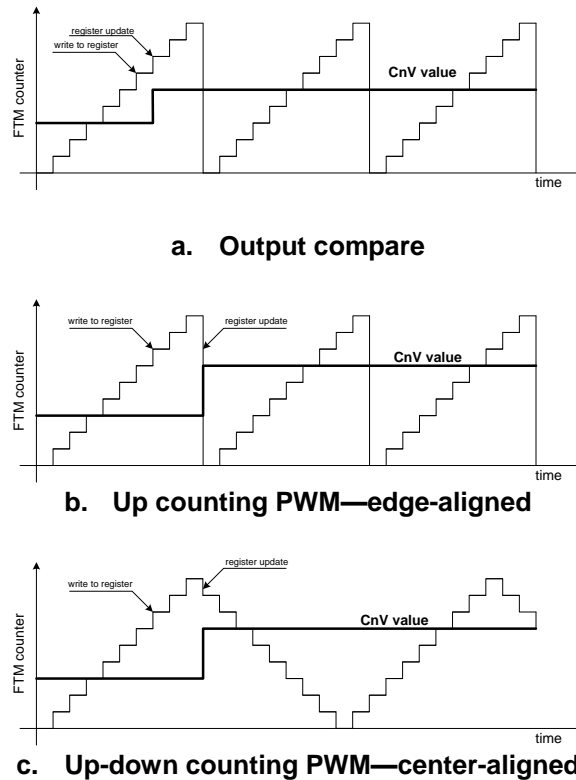


Figure 3. CnV register update when FTMEN = 0

## 5. FTM Approach (FTMEN = 1)

The previously described properties of the FTM module are inherited from the TMP module, which is used in the low-end devices (such as Kinetis L or HCS08). A very useful feature of the FTM is updating the buffered register at a certain time (different from the previously mentioned methods). You can do this using the software or hardware triggers (another peripheral or FTM module can trigger the register update).

### NOTE

The software triggering takes place when a value of 1 is written to the SWSYNCH bit in the FTMx\_SYNC register. The SWSYNCH bit is cleared automatically, depending on the type of the synchronization event (SWSOC, SWINVC, SWOM, SWWRBUF, or SWRSTCNT). This takes one system clock cycle or one loading point event after writing to the register.

### NOTE

The register buffers are ignored when the FTM counter enters any of the BDM modes (except for the functional mode BDMODE = 0x3, see the specific reference manual for more details).

The PWM synchronization feature is often required in more complex motor-control/power-conversion applications. It is divided into two mechanisms called legacy and enhanced PWM synchronization.

It is recommended to use only the enhanced PWM synchronization, because it also includes all features of the legacy PWM synchronization.

### NOTE

It is expected that this approach for the register update is used only with the combine PWM mode.

## 5.1. Legacy PWM synchronization (SYNCMODE = 0)

The legacy PWM synchronization (SYNCMODE = 0) represents a subset of the enhanced PWM synchronization (SYNCMODE = 1) and you can use it only with a reduced set of buffered registers (MOD, CnV, and OUTMASK).

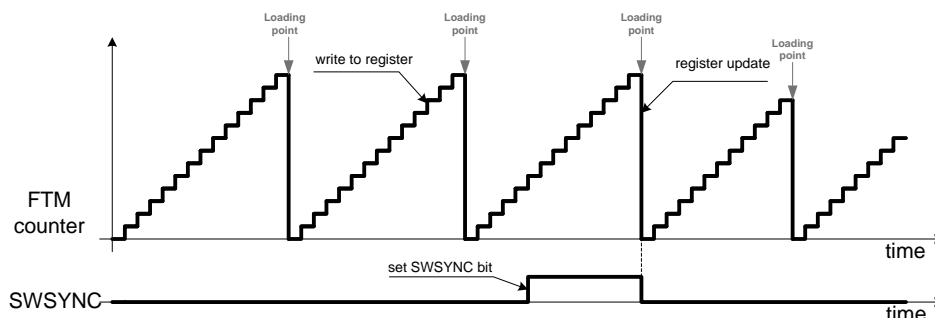
The additional feature of the legacy PWM synchronization is the possibility to choose between two options of trigger restriction (the PWMSYNC bit in the FTMx\_MODE register). Use both the software and hardware triggers to update the MOD, CnV, and OUTMASK registers when the PWMSYNC bit is 0. Use the software trigger for the MOD and CnV registers, and use the hardware trigger for the OUTMASK register only when the PWMSYNC bit is 1.

The MOD register update using the legacy PWM synchronization is shown in [Figure 4](#) (with the software trigger) and in [Figure 5](#) (with the hardware trigger). The loading points in the figures represent the MOD value.

The legacy PWM synchronization also supports the FTM counter re-initialization. This feature is available via the REINIT bit in the FTMx\_SYNC register. When the REINIT bit is set, then the register update occurs in the next system clock cycle after the trigger event (see [Figure 6](#) and [Figure 7](#)).

The principle of register update for the MOD register (shown in [Figure 4](#) and [Figure 5](#)) is also valid for the CnV register.

The detailed description of the OUTMASK register update is shown in the following sections. Consider the trigger restriction (PWMSYNC bit) in case of the legacy PWM synchronization.



**Figure 4. MOD register update with legacy PWM synchronization—software trigger**

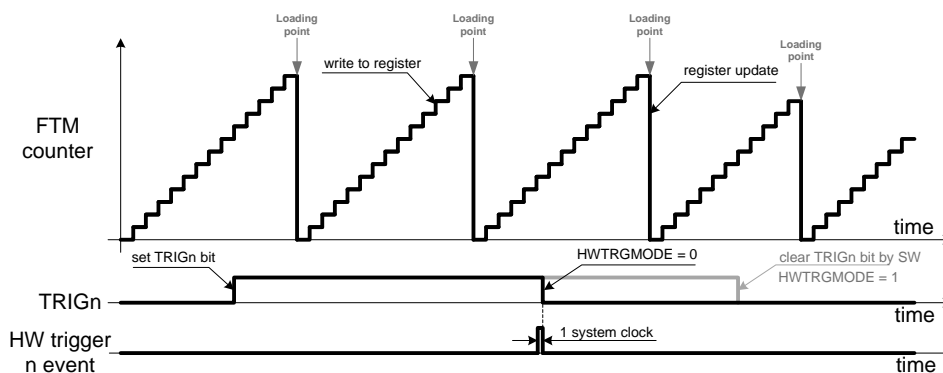


Figure 5. MOD register update with legacy PWM synchronization—hardware trigger

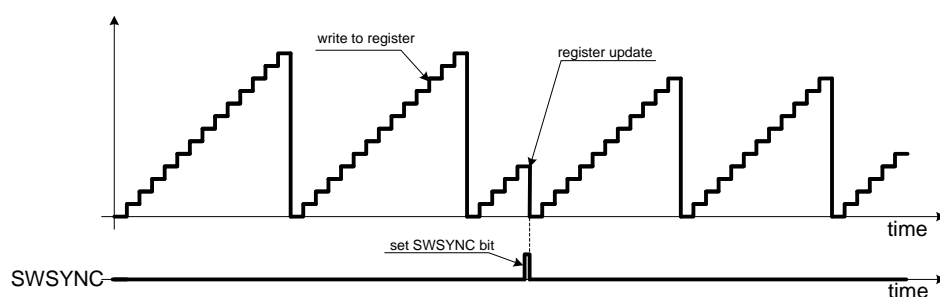


Figure 6. MOD register update with legacy PWM synchronization—software trigger (REINIT = 1)

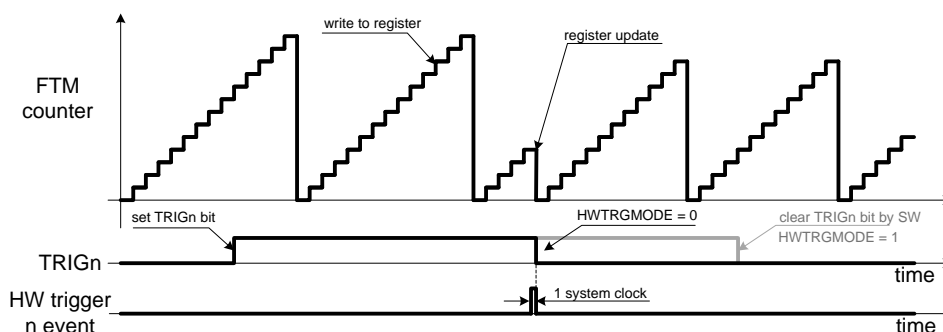


Figure 7. MOD register update with legacy PWM synchronization—hardware trigger (REINIT = 1)

### NOTE

Only the MOD, CnV, and OUTMASK registers can be used by the legacy PWM synchronization. However, it is expected that these registers are synchronized only by the enhanced PWM synchronization.

### NOTE

Do not use both the software and hardware triggers with the legacy PWM synchronization at the same time. Unpredictable behavior can occur when they are used together.

## 5.2. Enhanced PWM synchronization (SYNCMODE = 1)

The legacy PWM synchronization (SYNCMODE = 0) is a subset of the enhanced PWM synchronization (SYNCMODE = 1). Apply the enhanced PWM synchronization to each buffered register to perform a synchronization event—including the MOD, CNTIN, CnV, OUTMASK, SWOCTRL, and INVCTRL registers. Use the hardware or software triggers for a synchronized register update.

The MOD, CNTIN, and CnV registers use the same approach for register update. It is not possible to perform a separate register update. Only one bit (HWWRBUF or SWWRBUF) in the FTMx\_SYNCOF register enables updating all three registers (MOD, CNTIN, and CnV). All these registers (MOD, CNTIN, and CnV) are updated by the buffer value in the same synchronization event (when they are configured properly and a trigger event occurs).

The other registers (OUTMASK, SWOCTRL, and INVCTRL) use individual configuration bits (HWOM, HWINVC, HWSOC or SWOM, SWINVC, SWSOC) in the FTMx\_SYNCONF register to perform a separate register update.

The following subsections explain the PWM enhanced synchronization applied to each of the register updates in more detail.

### 5.2.1. MOD register update (SWWRBUF = 1 or HWWRBUF = 1)

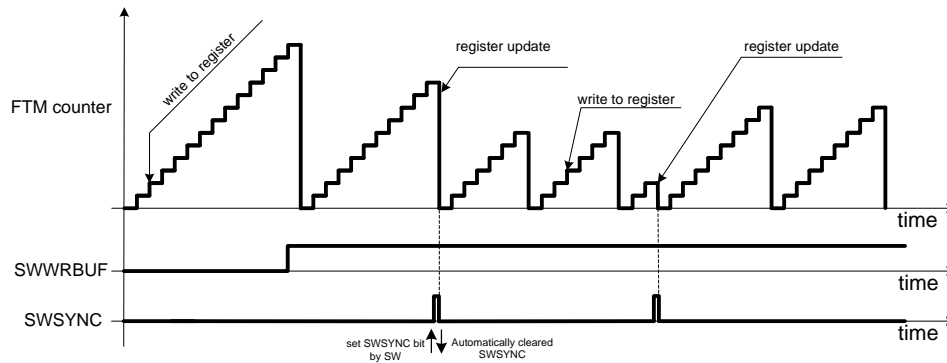
Set the SWWRBUF or HWWRBUF bits (depends on the hardware or software triggers used) in the FTMx\_SYNCONF register to update the MOD register with its buffer value using the enhanced PWM synchronization. The following figures (Figure 8 to Figure 13) show the time diagrams of the selected techniques of register update.

#### NOTE

The figures in this subsection consider that only the MOD register buffer is filled with a new value before the enhanced PWM synchronization event occurs.

#### 5.2.1.1. Software trigger (HWWRBUF = 0)

The following figure represents the time diagram of a situation when the software trigger with the counter reset enabled (SWRSTCNT bit is set) is used for the register update. In case the software trigger event is generated (by setting SWSYNC bit in the software), the register update occurs immediately after one system clock period. The counter is reset to its CNTIN value due to the SWRSTCNT being set. The SWSYNC bit is also cleared with the counter reset. The register is updated with its new value (if it is written to the register buffer) in the same way on each software trigger event when the SWWRBUF bit is set.



**Figure 8. MOD register update with enhanced PWM synchronization—software trigger (SWRSTCNT = 1)**

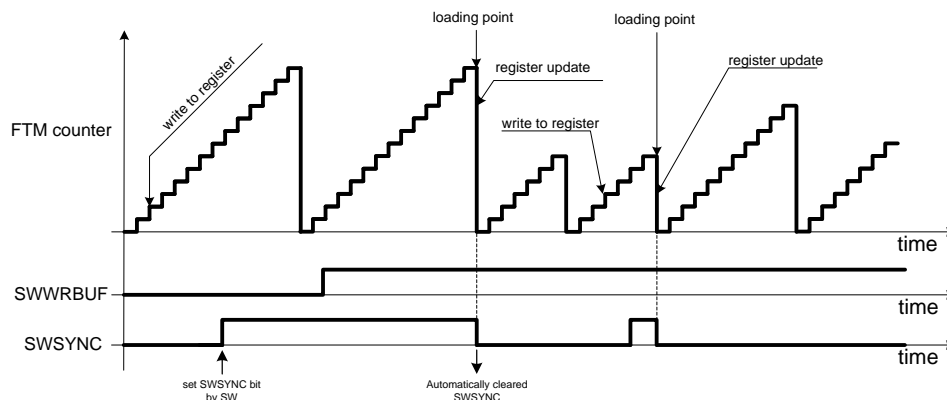
Figure 9 represents the time diagram of a situation when the software trigger is used but the counter reset is disabled (the SWRSTCNT bit is cleared). In case the software trigger event is generated (by setting the SWSYNC bit in software), the register update occurs after the counter reaches the loading point. The SWSYNC bit is cleared with the register update.

### CAUTION

The enhanced PWM synchronization event is not successful when the SWSYNC bit is cleared by software before the counter reaches the loading point.

### NOTE

Considering that only the combine mode is used with the enhanced PWM synchronization, the loading point may only be of the MOD value.



**Figure 9. MOD register update with enhanced PWM synchronization—software trigger (SWRSTCNT = 0)**

#### 5.2.1.2. Hardware trigger (HWRBUF = 1)

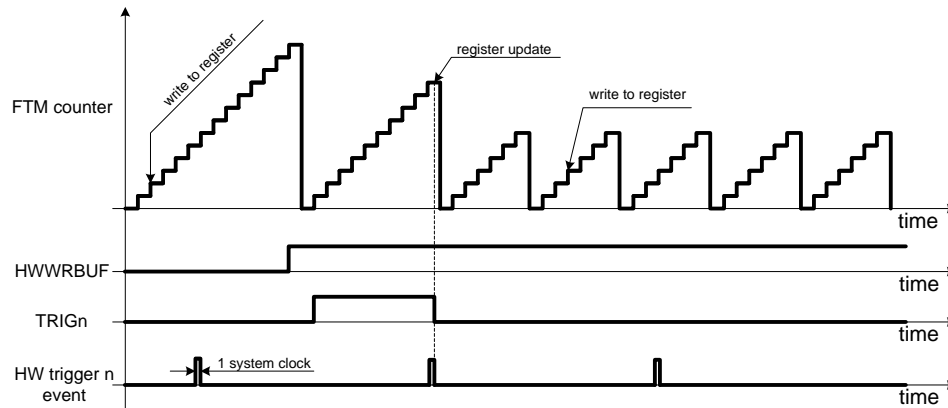
The time diagram shown in Figure 10 represents the MOD register update using hardware trigger with the counter reset enabled (the HWRSTCNT bit is set) and the hardware trigger mode disabled (HWTRGMODE bit is cleared). Figure 10 shows that the counter is reset and the MOD register is updated by its buffer value one system clock cycle after the hardware trigger event is detected.



When the next hardware trigger n event occurs, the counter is not reset and the MOD register is not updated. This happens because the HWTRIGMODE is disabled and the TRIGn bit is automatically cleared one system clock cycle after the hardware trigger n event is detected. In such case, set the TRIGn bit by software to update the register with its buffer value again.

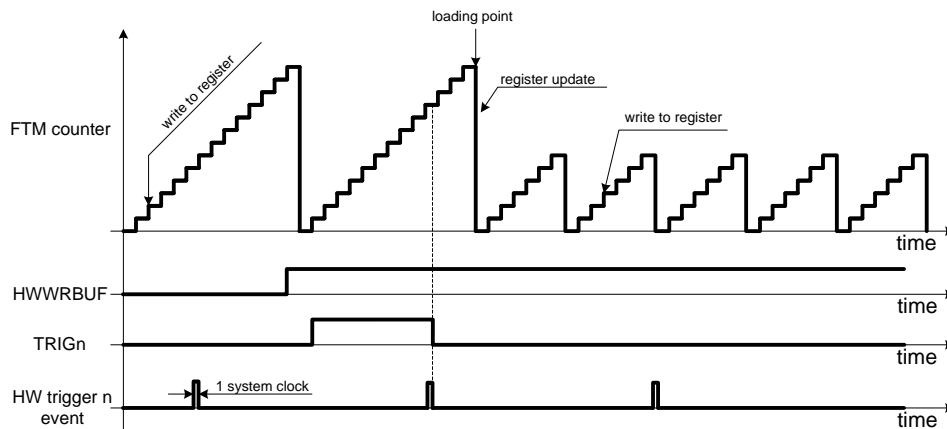
### NOTE

The hardware trigger mode bit HWTRIGMODE affects the TRIGn bits after the hardware trigger n event occurs. When the HWTRIGMODE is set, then the TRIGn bits remain in their states. When the HWTRIGMODE is 0, then the TRIGn bit is cleared one system clock cycle after the hardware trigger n event is detected.



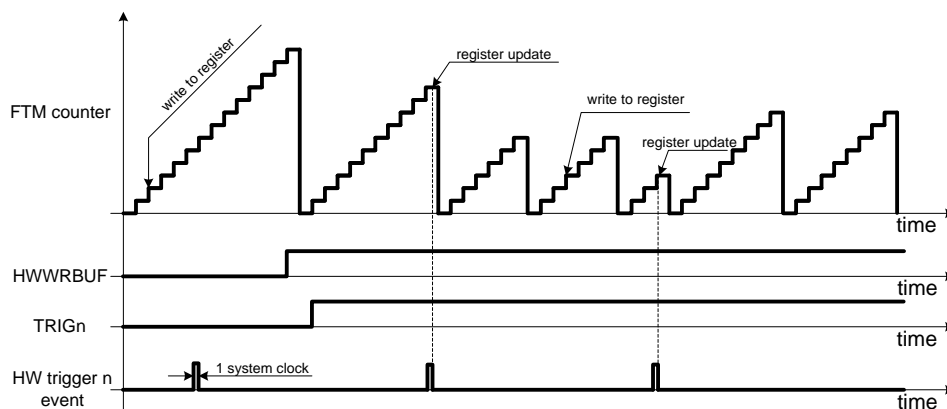
**Figure 10. MOD register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, HWRSTCNT = 1)**

The following figure shows a similar case, but with the counter reset disabled. In this case, the counter is not reset, and the register is updated when the counter reaches the loading point after the hardware trigger n event is detected.



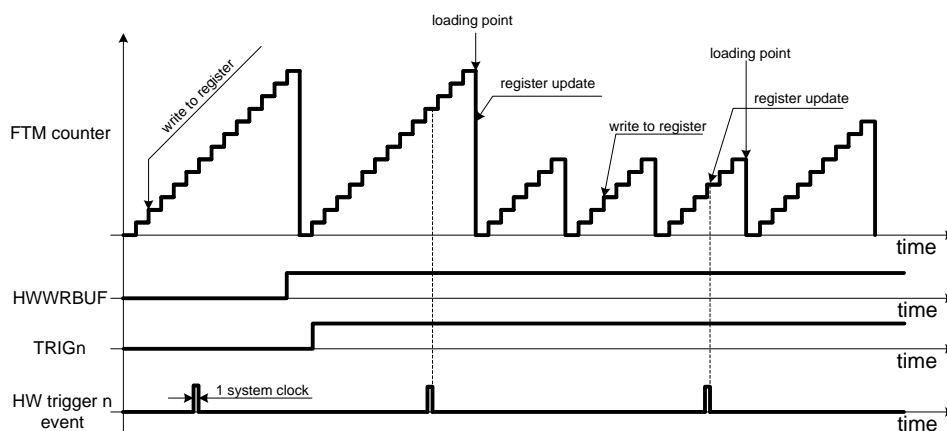
**Figure 11. MOD register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, HWRSTCNT = 0)**

The following figure shows the case when the hardware trigger is used with the counter reset and the HWTRIGMODE enabled. In such case, the counter is reset and the MOD register is updated one system clock cycle after the hardware trigger n event is detected. There is no need to set the TRIGN bit by software, because the HWTRIGMODE is enabled. The TRIGN bit stays set until it is cleared by software. The MOD register update is enabled with each subsequent hardware trigger n event.



**Figure 12. MOD register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, HWRSTCNT = 1)**

The following figure shows an example similar to the previous examples, except for the counter reset. The MOD register is updated at the loading point, after the hardware trigger n event is detected.



**Figure 13. MOD register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, HWRSTCNT = 0)**

### 5.2.2. CNTIN register update (HWWRBUFF = 1 or SWWRBUFF = 1)

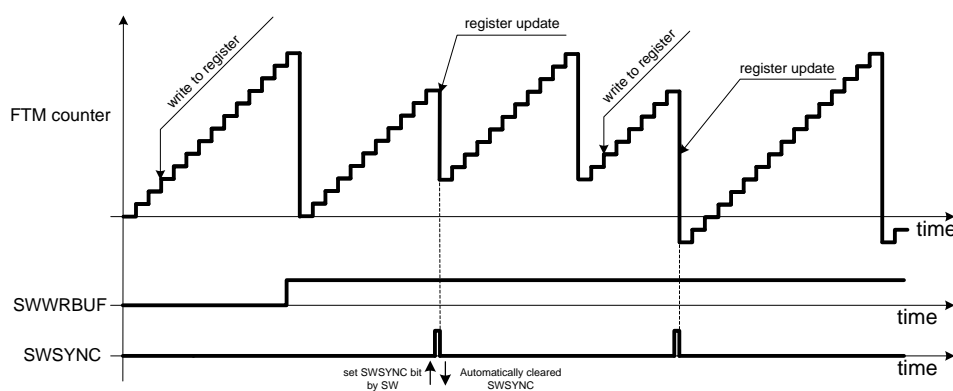
This register update is similar to the MOD register update. Set the SWWRBUFF or HWWRBUFF bits (depending on the hardware or software trigger used) in the FTMx\_SYNCONF as in the case of the MOD register. The figures from Figure 14 to Figure 19 show the time diagrams of the selected techniques of the CNTIN register update. The figures from Figure 14 to Figure 19 are very similar to the figures from Figure 8 to Figure 13.

**NOTE**

The figures in the following subsections consider that only the CNTIN register buffer is filled with the new value before the enhanced PWM synchronization event occurs.

**5.2.2.1. Software trigger (SWWRBUF = 1)**

The following figure shows the time diagram of using the software trigger with the counter reset enabled (SWRSTCNT bit is set) to update the register. In such case, the CNTIN register update occurs one system clock cycle after the software trigger event is generated (setting the SWSYNC bit by software). The counter is reset to its updated CNTIN value because the SWRSTCNT is set. The SWSYNC bit is also cleared on the counter reset.

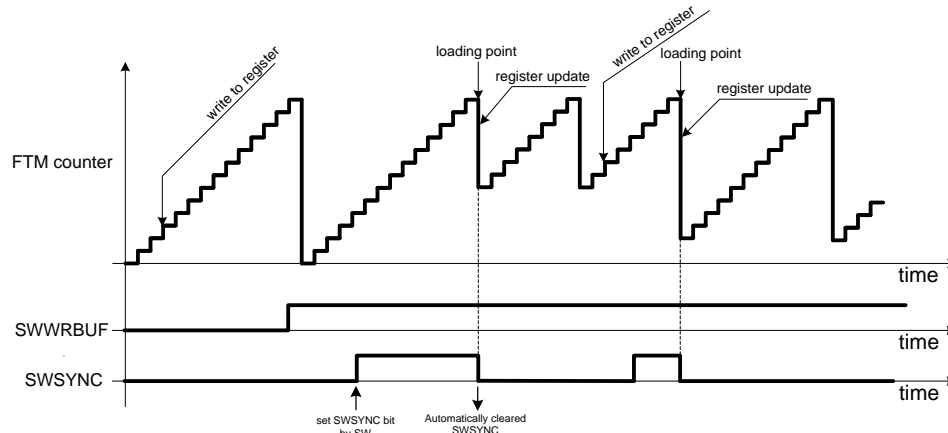


**Figure 14. CNTIN register update with enhanced PWM synchronization—software trigger (SWRSTCNT = 1)**

Figure 15 shows the time diagram of a situation when the software trigger is used but the counter reset is disabled (the SWRSTCNT bit is cleared). In such case, the CNTIN register is updated by its buffer value at the loading point, after the software trigger event is detected. The SWSYNC bit is cleared on the register update.

**NOTE**

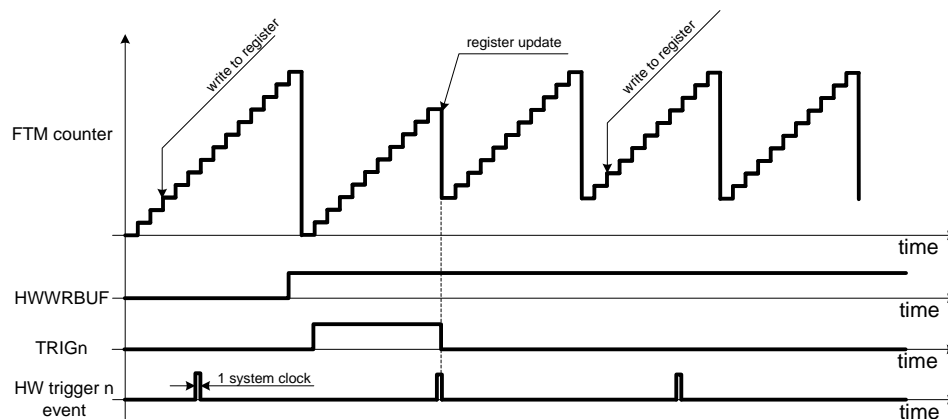
Considering that only the combine mode is used with the enhanced PWM synchronization, the loading point can only have the MOD value.



**Figure 15. CNTIN register update with enhanced PWM synchronization—software trigger (SWRSTCNT = 0)**

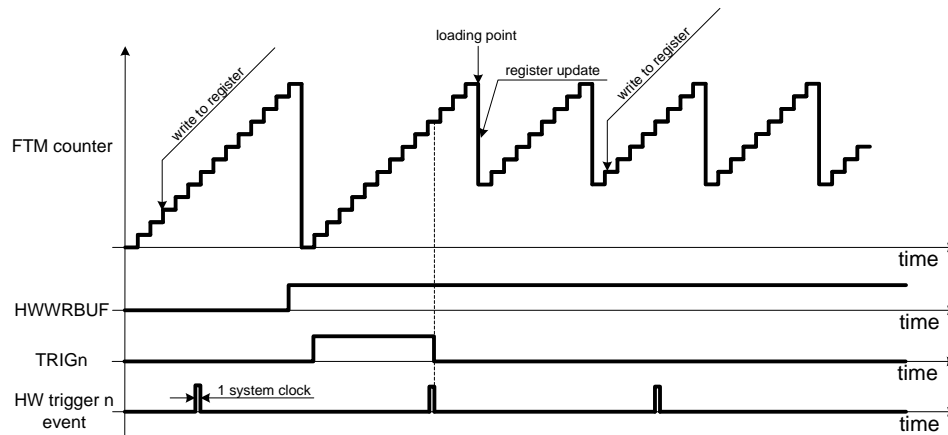
### 5.2.2.2. Hardware trigger (HWRSTCNT = 1)

The time diagram in the following figure shows the CNTIN register update using the hardware trigger with the counter reset enabled (the HWRSTCNT bit is set) and the hardware trigger mode disabled (the HWTRGMODE bit is cleared). The following figure shows that the counter is reset and the CNTIN register is updated with its buffer value one system clock cycle after the hardware trigger event is detected. When the next hardware trigger event occurs, the counter is not reset and the MOD register is not updated. This happens because the HWTRIGMODE is disabled and the TRIGN bit is automatically cleared one system clock cycle after the hardware trigger event is detected. In such case, set the TRIGN bit by software to update the register with its buffer value again.



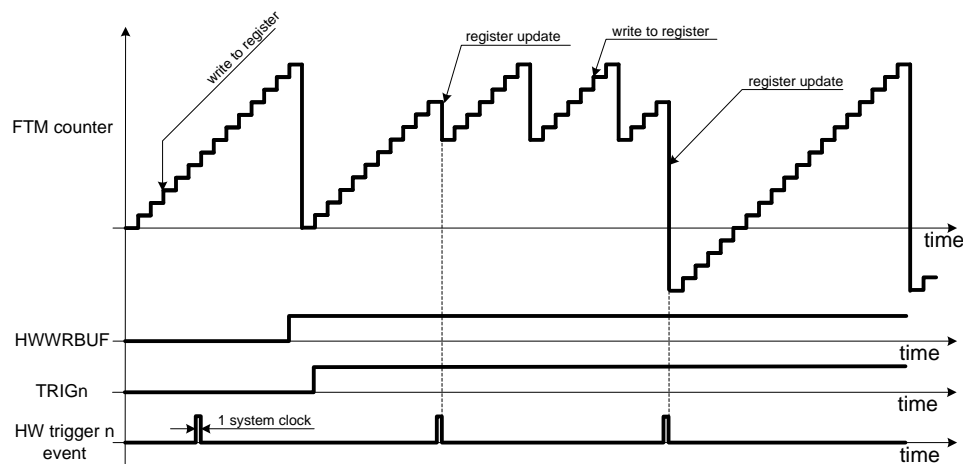
**Figure 16. CNTIN register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, HWRSTCNT = 1)**

The following figure shows a similar case, but the counter reset is disabled. In this case, the counter is not reset and the register is updated when the counter reaches the loading point after the hardware trigger event is detected.



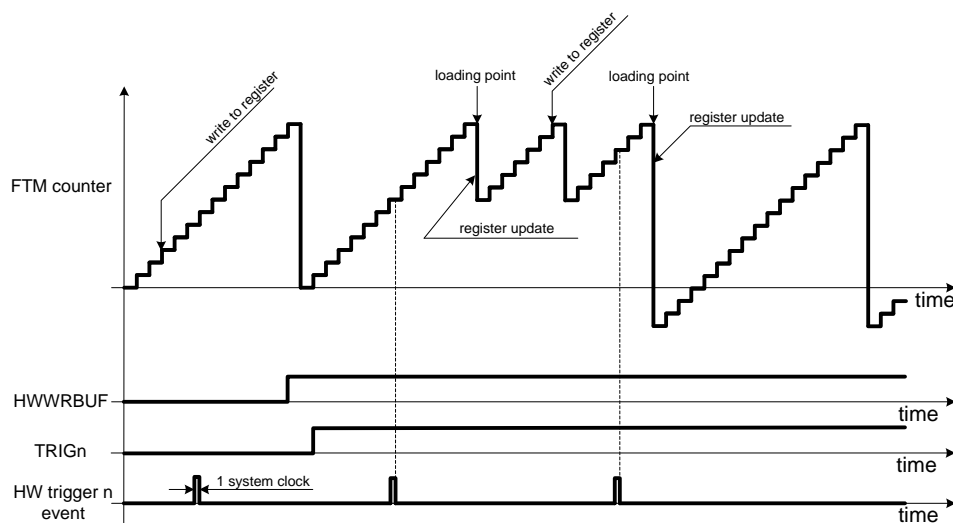
**Figure 17. CNTIN register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, HWRSTCNT = 1)**

The following figure shows the case when the hardware trigger is used with the counter reset and the HWTRIGMODE enabled. In such case, the counter is reset and the MOD register is updated one system clock cycle after the hardware trigger n event is detected. You don't have to set the TRIGn bit by software, because the HWTRIGMODE is enabled. The TRIGn bit stays set until it is cleared by software. The MOD register update is enabled with each subsequent hardware trigger n event.



**Figure 18. CNTIN register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, HWRSTCNT = 1)**

The following figure shows an example similar to the previous one, except for the counter reset. The MOD register is updated at the loading point after the hardware trigger n event is detected.



**Figure 19. CNTIN register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, HWRSTCNT = 0)**

### 5.2.3. CnV register update (HWWRBUF = 1 or SWWRBUF = 1)

This register update has the same manner as the MOD register update. Set the SWWRBUF or HWWRBUF bits (depends on the hardware or software trigger used) in FTMx\_SYNCON. The figures from [Figure 20](#) to [Figure 25](#) show the time diagram of the selected techniques of the CnV register update. There is a high degree of similarity between figures from [Figure 20](#) to [Figure 25](#) and figures from [Figure 8](#) to [Figure 13](#).

#### NOTE

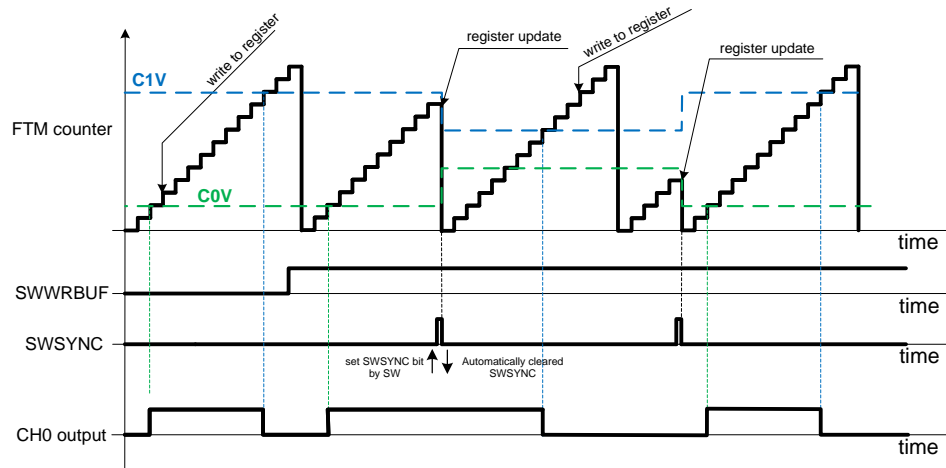
The figures in this subsection consider that only the CnV register buffer is filled with the new value before the enhanced PWM synchronization event occurs. The FTM module is configured to work in the combine mode.

#### 5.2.3.1. Software trigger (SWWRBUF = 1)

[Figure 20](#) shows the time diagram of a situation when the software trigger with the counter reset enabled (the SWRSTCNT bit is set) is used to update the register. In such case, the CnV register update occurs one system clock cycle after the software trigger event is generated (setting the SWSYNC bit by software). The counter is reset to its CNTIN value because the SWRSTCNT bit is set. The SWSYNC bit is also cleared with the counter reset.

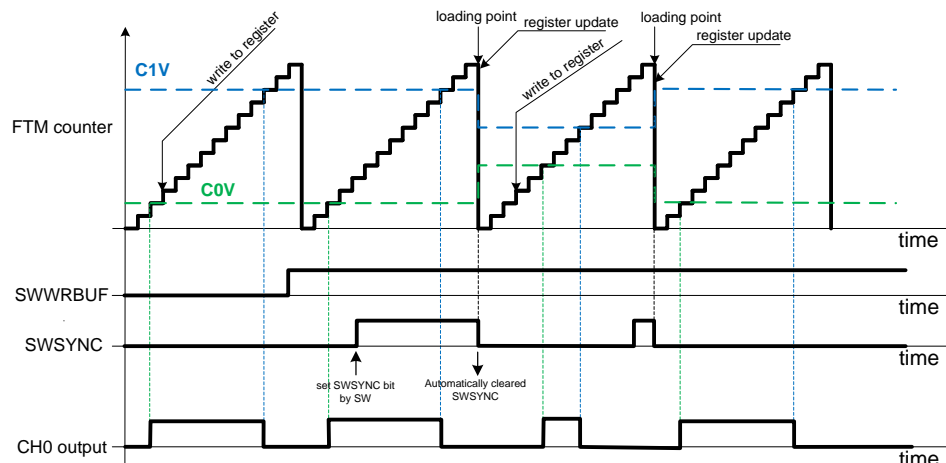
## NOTE

Notice that the SWRSTCNT (or HWRSTCNT) bit can affect the resulting duty cycle of the PWM signal. When the software (or hardware) trigger event occurs before the compare event ( $CNT = CnV$  or  $CNT = C(n+1)V$ ), the counter is reset but the rising/falling edge (depending on the ELSx bits' configuration as well as on the  $CnV/C(n+1)V$  compare event) does not appear (see Figure 20, Figure 22, and Figure 24).



**Figure 20. CnV register update with enhanced PWM synchronization—software trigger (SWRSTCNT = 1)**

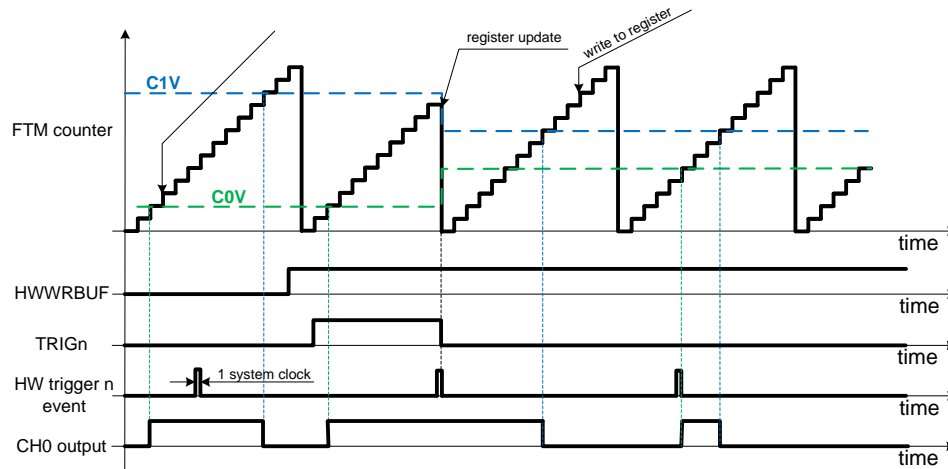
Figure 15 shows the time diagram of a situation when the software trigger is used, but the counter reset is disabled (the SWRSTCNT bit is cleared). In such case, the  $CnV$  and  $C(n+1)$  registers are updated with their buffer values at the loading point after the software trigger event is detected. The SWSYNC bit is cleared with the register update.



**Figure 21. CnV register update with enhanced PWM synchronization—software trigger (SWRSTCNT = 0)**

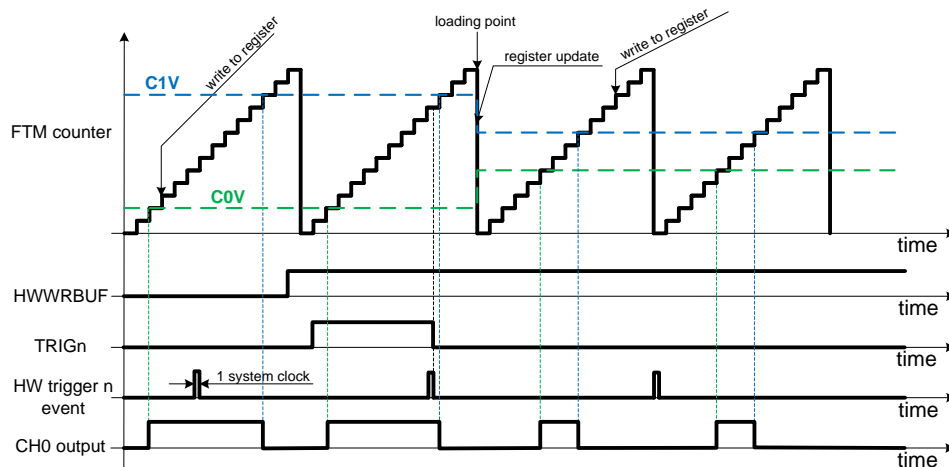
### 5.2.3.2. Hardware trigger (HWRBUF = 1)

The time diagram in the following figure shows a situation where the CnV register is updated using a hardware trigger with the counter reset enabled (the HWRSTCNT bit is set) and the hardware trigger mode disabled (the HWTRIGMODE bit is cleared). The following figure shows that the counter is reset and the CnV/C(n+1)V registers are updated with their buffer values one system clock after the hardware trigger event is detected. When the next hardware trigger event occurs, the counter is not reset and the CnV/C(n+1)V registers are not updated. This happens because the HWTRIGMODE is disabled and the TRIGN bit is automatically cleared one system clock cycle after the hardware trigger event is detected. In such case, set the TRIGN bit by software to update the register with its buffer value.



**Figure 22. CnV register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, HWRSTCNT = 1)**

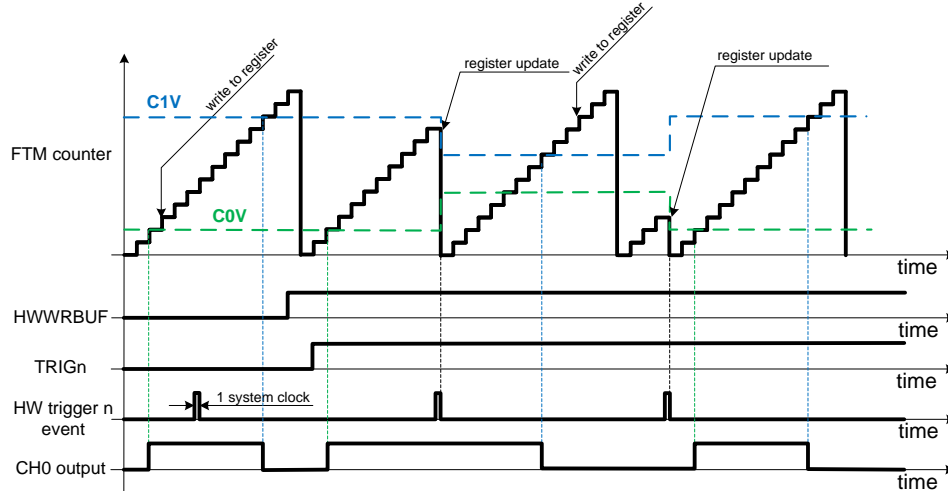
The following figure shows a similar case, but the counter reset is disabled. In this case, the counter is not reset and the registers are updated when the counter reaches the loading point after the hardware trigger event is detected.



**Figure 23. CnV register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, HWRSTCNT = 0)**

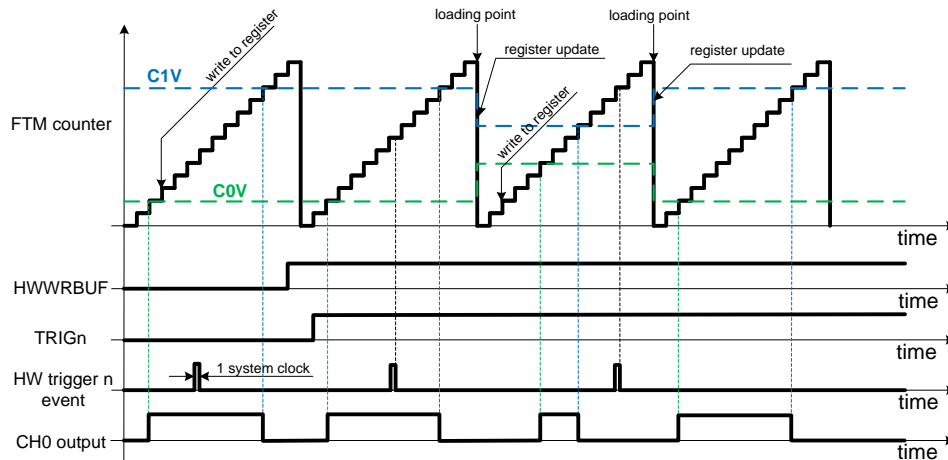


The following figure shows a case where the hardware trigger is used with the counter reset enabled and the HWTRIGMODE enabled. In such case, the counter is reset and the CnV/C(n+1)V registers are updated one system clock cycle after the hardware trigger event is detected. There is no need to set the TRIGn bit by software, because the HWTRIGMODE is enabled. The TRIGn bit stays set until it is cleared by software. The CnV registers update is enabled with each subsequent hardware trigger event.



**Figure 24. CnV register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, HWRSTCNT = 1)**

The following figure shows an example similar to the previous one, except for the counter reset. The CnV registers are updated at the loading point, after the hardware trigger event is detected.



**Figure 25. CnV register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, HWRSTCNT = 0)**

## 5.2.4. OUTMASK register update (SYNCHOM = 1)

Set the SYNCHOM bit in the FTMx\_SYNC register to update the OUTMASK register using enhanced PWM synchronization. Use either the software or hardware triggers.

The register update takes place one system clock cycle after the software/hardware trigger is generated, as shown in Figure 26 and Figure 27. When a specific channel bit of the OUTMASK register is set to 1, then the corresponding FTM channel output is forced to the state defined by the channel initial state in the OUTINIT register (safe state).

### 5.2.4.1. Software trigger (SWOM = 1)

The following figure represents the time diagram of the OUTMASK register update using the software trigger. As shown in the following figure, the first software trigger generation does not cause the OUTMASK register update. The update happens because the SWOM bit is still 0. If the SYNCHOM and SWOM bits are already set, then the OUTMASK register update occurs one system clock cycle after the software trigger event is generated (setting the SWSYNC bit by software). The next update is generated by software (writing 1 to the SWSYNC bit) after the SWSYNC bit is cleared (either automatically at the loading point, or manually by software). The gray area in this figure shows the interval between the register updates (the register buffer value changes before both update events).

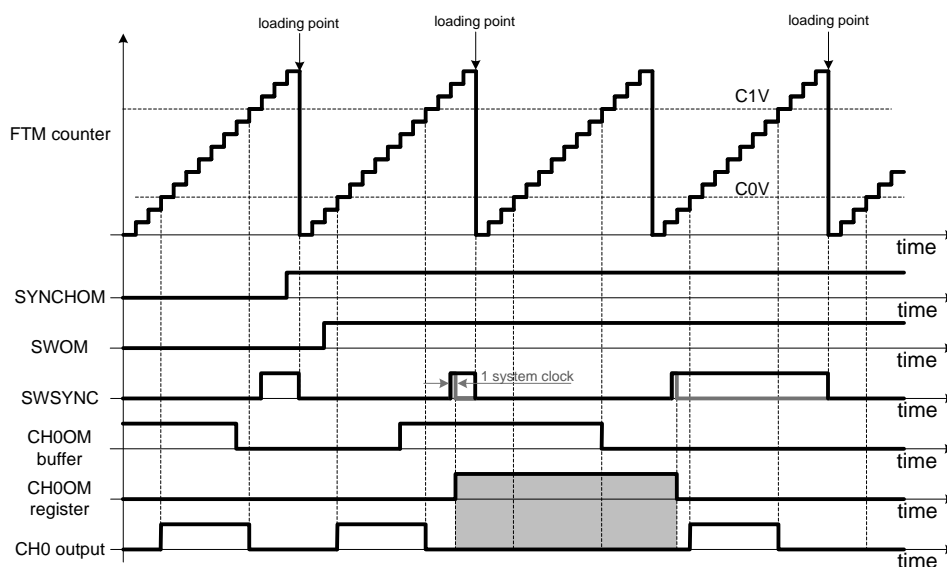


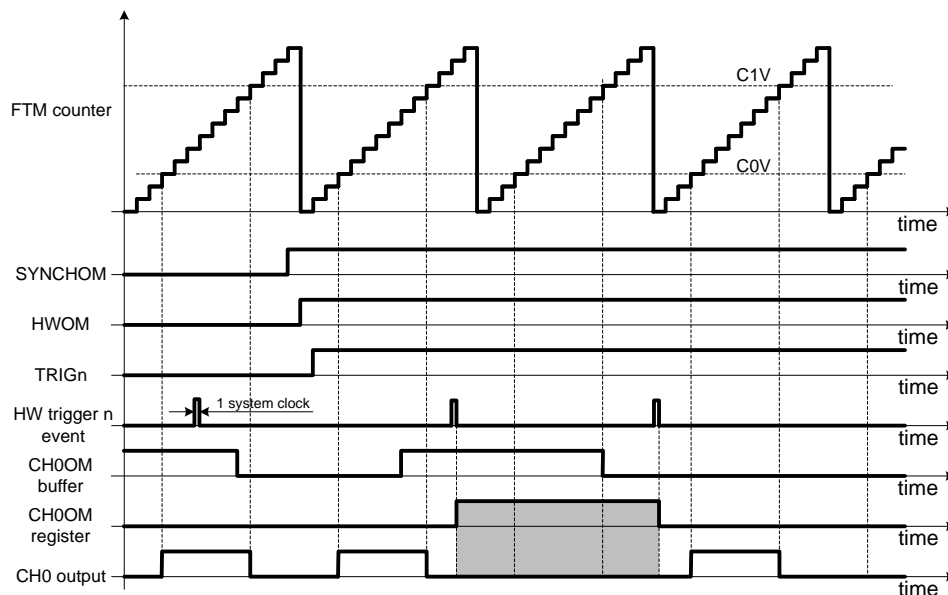
Figure 26. OUTMASK register update with enhanced PWM synchronization—software trigger (CH0OI = 0)

### 5.2.4.2. Hardware trigger (HWOM = 1)

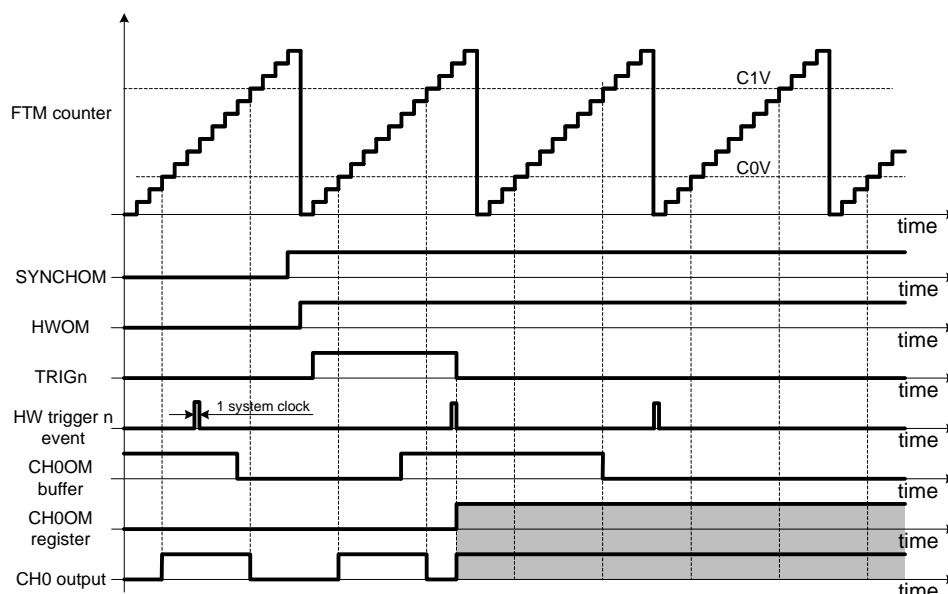
Figure 27 and Figure 28 show the time diagrams of the OUTMASK register update using the hardware trigger. If both the SYNCOM and the HWOM bits are set, the OUTMASK register update occurs one system clock cycle after the hardware trigger n is generated. According to the hardware trigger mode selection, the TRIGN bit is cleared (HWTRIGMODE = 0) on the register update or remains set (HWTRIGMODE = 1). If cleared (see Figure 28), set it by software to generate the register update event. If it remains set (see Figure 27), the next register update occurs when the new hardware trigger n event is detected. Figure 27 and Figure 28 also show the output behavior for different initial states of the channels' configuration.

In the case shown in Figure 27, the initial state for the corresponding channel output is set to a low level. The output of the corresponding channel is forced to a low level when the corresponding bit in the OUTMASK register (not buffer) is set.

In the case shown in Figure 28, the initial state for the output of the corresponding channel is set to a high level. The corresponding channel output is forced to a high level when the corresponding bit in the OUTMASK register is set.



**Figure 27. OUTMASK register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1, CH0OI = 0)**



**Figure 28. OUTMASK register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0, CH0OI = 1)**

### 5.2.5. INVCTRL register update (INVC = 1)

Set the INVC bit in the FTMx\_SYNC register to update the INVCTRL register using the enhanced PWM synchronization. Use either the software or hardware triggers.

The register update takes place one system clock cycle after the software/hardware trigger generation, as shown in figures from [Figure 29](#) to [Figure 31](#). When a specific channel pair bit of the INVCTRL register is set, then the corresponding FTM channel pair outputs are forced to their inverted states (e.g., channel 0 output becomes channel 1 output and contrariwise).

#### 5.2.5.1. Software trigger (SWINVC = 1)

The following figure shows the time diagram of the INVCTRL register update using the software trigger. The first software trigger generation does not cause the INVCTRL register update. It happens because the SWINVC bit is still 0. If the INVC and SWINVC bits are already set, then the INVCTRL register update occurs one system clock after the software trigger event is generated (setting the SWSYNC bit by software). The next update can be generated by software (writing 1 to SWSYNC) after the SWSYNC bit is cleared (either automatically at the loading point or manually by software). The gray area in the following figure shows the interval between two register updates (the register buffer value is changed before both update events).

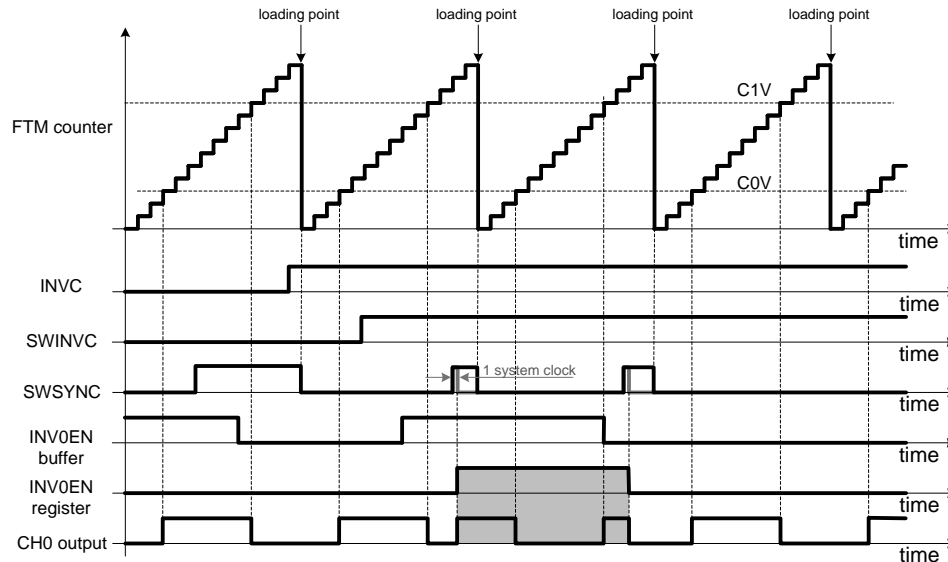
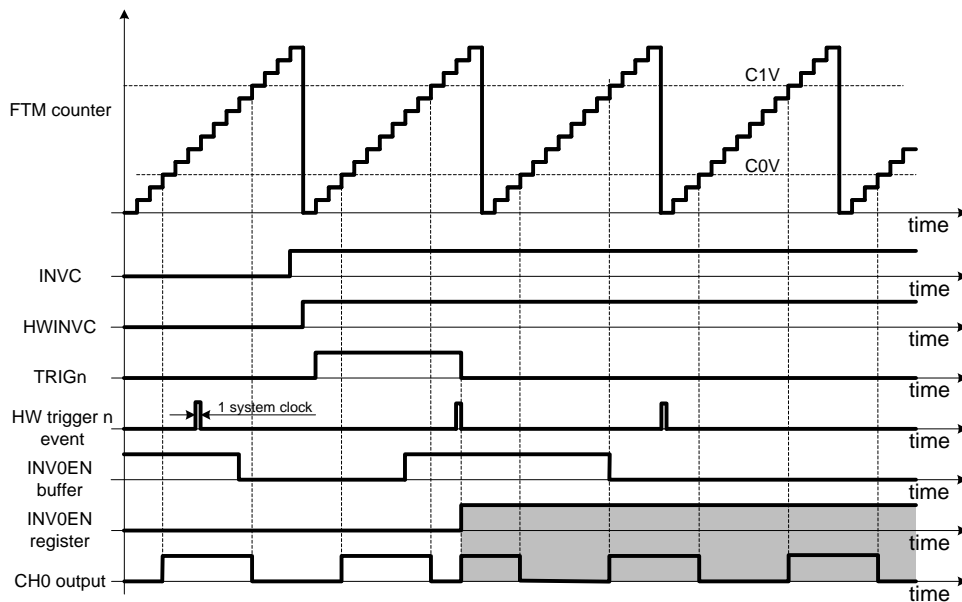


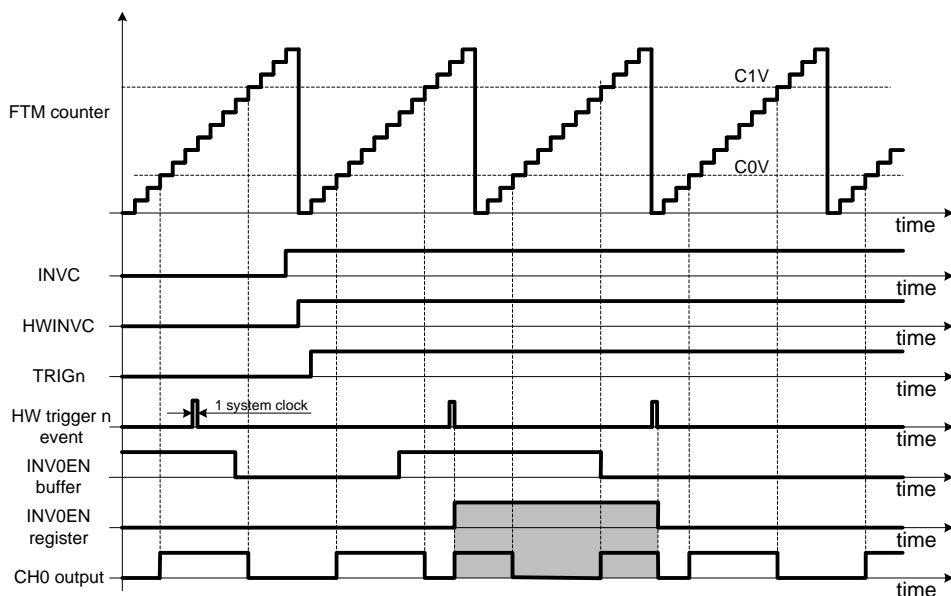
Figure 29. INVCTRL register update with enhanced PWM synchronization—software trigger

#### 5.2.5.2. Hardware trigger (HWINVC = 1)

The following two figures represent the time diagrams of the INVCTRL register update using hardware trigger. If both the INVC and HWINVC bits are set, then the INVCTRL register update occurs one system clock cycle after the hardware trigger  $n$  is generated. According to the hardware trigger mode selection, the TRIGN bit is cleared (HWTRIGMODE = 0) with the register update or remains set (HWTRIGMODE = 1). If it is cleared (see [Figure 30](#)), set it by software to generate the register update event. If it remains set (see [Figure 31](#)), then the next register update occurs when a new hardware trigger  $n$  event is detected. The following two figures also show the output behavior when the INVCTRL register is updated. The gray area in both figures shows the interval between two register updates (the register buffer value is changed before both update events).



**Figure 30. INVCTRL register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 0)**



**Figure 31. INVCTRL register update with enhanced PWM synchronization—hardware trigger (HWTRIGMODE = 1)**

## 6. Read Coherency Mechanism used in Dual Edge Capture Mode (FTMEN = 1, DECAPEN = 1)

A special case of the CnV registers update is represented by the FTM module working in the dual edge capture mode. The dual edge capture mode enables the capture buffers and the read coherency mechanism. It is applied to the pulse width as well as to the period measurement, which is in the dual edge capture mode of the FTM. The read coherency mechanism avoids data loss in the special cases of measuring the pulse width or period. The basic principle of the read coherency mechanism is shown in Figure 21. It includes these three important timelines:

1. The first edge detection on the channel n input pin—the current counter value is transferred to the CnV buffer.
2. The second edge detection on the channel n input pin—the current counter value is transferred to the C(n+1)V buffer and the buffer value of CnV is transferred to the CnV register.
3. The previous two stages are done automatically by hardware. The third stage requires software intervention. When the CnV register is read by software, the value of the C(n+1)V buffer is transferred to the C(n+1)V register. After that, the pulse width/period is calculated as a subtraction of the C(n+1) register value from the CnV register value. Multiply the subtraction by the FTM counter clock period to get the direct time value (in seconds).

### NOTE

If the C(n+1)V register is read before the CnV register, then an incorrect pulse width/period is measured.

### NOTE

The type of the first edge and the second edge is dependent on the configuration, pulse width (positive or negative), or period measurement.

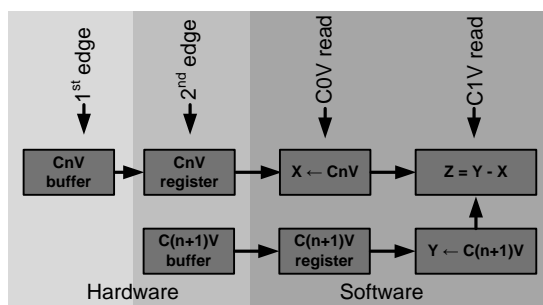


Figure 32. Read coherency mechanism principle

Figure 32 and Figure 33 show two special cases of the positive pulse width measurement that you may encounter. The same analogy is found in the case of the negative pulse width/period measurement.

The first case (shown in Figure 33) describes a situation where the second rising edge (there are three edges in total) occurs before the software routine is called. In this case, there is no data loss because the second rising edge (third edge in total) is captured into the buffer and not into the register. The pulse width measured during the previous FTM counter period is measured correctly.

In the software routine, the CnV register is read first (the value of the C(n+1)V buffer is transferred to the C(n+1)V register) and the C(n+1)V register is read after that. The pulse width (the gray area in Figure 33) is then calculated.

The second case (shown in Figure 34) represents a special case where the second falling edge (there are four edges in total) occurs before the software routine is called. In this case, the rising edge value (first edge) of the previous FTM counter period is lost, because the CnV register is already rewritten by its buffer value (the last rising edge—third edge) during the falling edge (fourth edge) detection. The pulse width that occurred during the previous FTM counter period is completely lost. In the software routine, the CnV register is read first (the value of second falling edge C(n+1)V buffer is transferred to the C(n+1)V register), followed by the C(n+1)V register. The pulse width occurs in the FTM counter overflow and the software routine call is calculated (the gray area in Figure 34).

### NOTE

The software routine represents a service routine where the CnV and C(n+1)V registers are read. This routine can also be the FTM timer overflow interrupt service routine. In such case, entering the routine is quite fast. However, consider the several core clock cycles (depends on the core used—CM4/CM0+, the interrupt priority, and the density of code before the register read) between the interrupt request and the register filling.

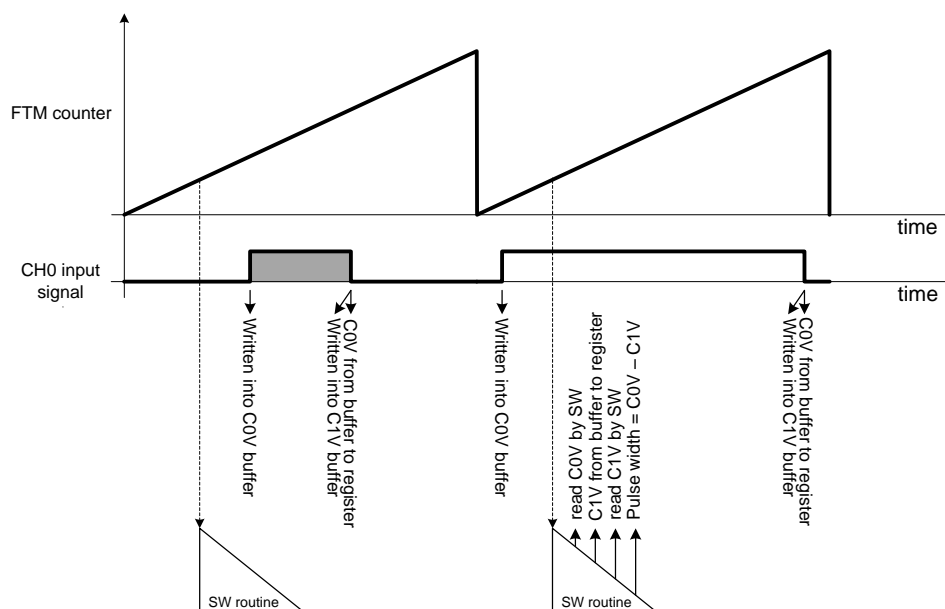


Figure 33. Read coherency mechanism—three edges detected before read in software routine



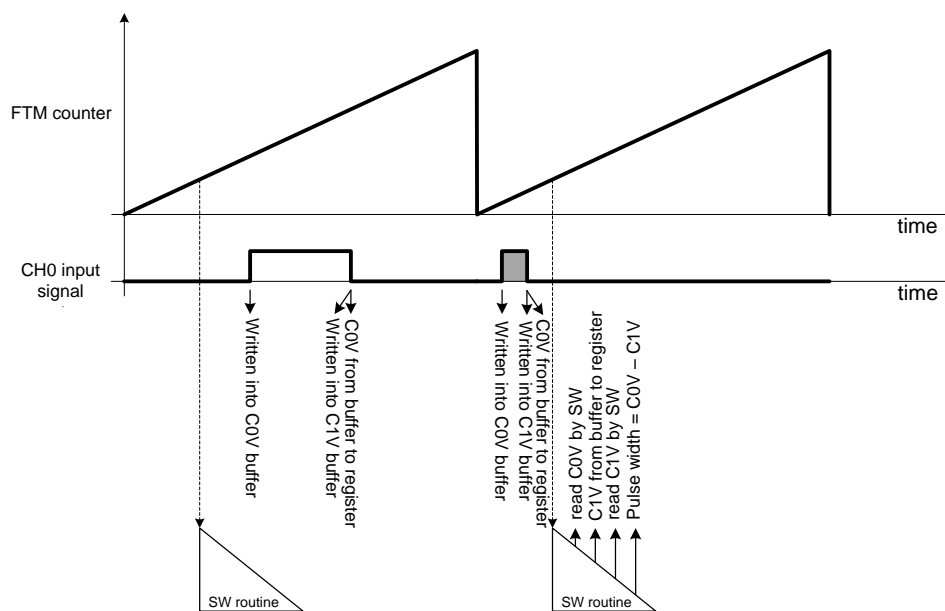


Figure 34. Read coherency mechanism—four edges detected before read in software routine

## 7. Demonstration Using Sample Code Example

This section describes the sample code which represents an integral part of this application note. This code example demonstrates the FTM module buffered register update feature on the background of application debugging (using FreeMASTER tool). It includes the three previously mentioned approaches of updating the buffered registers with their buffer values. The FRDM-K64F development board is used for the demonstration. The code is written using KSDK 1.3.

### 7.1. Peripherals configuration

The most important peripherals are configured in the code example as follows:

- Clock configuration:
  - Core/System = 120 MHz
  - Peripheral bus = 60 MHz
  - Flexbus = 60 MHz
  - Flash = 24 MHz
- FTM0 module (represents the FTM module on which the buffered register update is performed):
  - Clock source = peripheral bus clock = 60 MHz
  - Initial counter value CNTIN = -3000
  - Counter modulo MOD = 3000
  - PWM frequency =  $60 \text{ MHz} / 6000 = 10 \text{ kHz}$
  - Counter overflow interrupt is enabled (the PTC17 pin is toggled to symbolize the timer boundary cycles)

- Counter channels 0 and 1 are configured according to the selected register update approach (based on the FTM module configuration structure initialization *ftm0\_config*):
  - *FTM0\_TPM\_APPROACH\_CONFIG*—independent PWM up-counting mode with channel 0 compare values set to -1500 (25 % of the pulse width is placed on the counter start) and the channel is set to inverted 1500 (25 % of the pulse width is placed on the counter overflow)
  - *FTM0\_LEGACY\_PWM\_SYNC\_CONFIG* or *FTM0\_ENHANCED\_PWM\_SYNC\_CONFIG*—combined mode with complementary outputs, channels are set to 50 % of the duty
- Channel interrupts are disabled
- The FTM features related to the PWM synchronization are configured according to the selected register update approach (element *pwm\_sync\_config\_t* of the *ftm\_config\_t* FTM configuration structure)
  - *PWM\_SYNC\_DISABLE*—represents the TPM approach, no FTM features are enabled
  - *PWM\_SYNC\_LEGACY\_SWTRG\_HWTRG0\_CONFIG*:
    - Represents the legacy PWM synchronization with the CnV and MOD registers update available
    - Software or hardware triggers can be used
    - Counter re-initialization is not enabled by default (it can be enabled in runtime before calling the FTM0 initialization function)
  - *PWM\_SYNC\_ENHANCED\_SWTRG\_HWTRG0\_CONFIG*:
    - Represents enhanced PWM synchronization with the CNTIN, MOD, CnV, OUTMASK, SWOCTRL, and INVCTRL registers update available
    - Software or hardware triggers can be used
    - Counter restart (HWRSTCNT, SWRSTCNT) is not enabled by default (it can be enabled in runtime before calling the FTM0 initialization function)
    - Hardware trigger mode is disabled—this means that the TRIG0 bit is cleared when a hardware trigger is detected
- External triggers are not used
- *ftm0\_callback* function is called on each counter overflow
- PTC17 pin is toggled in the function (to symbolize the timer boundary cycles)
- FTM1 module (represents the hardware trigger 0 for the FTM0 module):
  - Clock source = peripheral bus / 1 = 60 MHz
  - Initial counter value CNTIN = 0
  - Counter modulo MOD = 12000 (twice the FTM0)
  - PWM frequency = 60 MHz / 12000 = 5 kHz (half of the FTM0)
  - Counter overflow interrupt is disabled
  - Channel 0 compare value is set to the modulo value (two periods on the FTM0)
  - Channel 0 interrupt is enabled

- FTM feature is disabled
- Channel 0 event is used as an external trigger connected to the FTM0 (as hardware TRIG0)
- *ftm1\_callback* function is called on a channel 0 event
  - A short pulse is generated on the PTC 16 pin (it represents the FTM0 hardware TRIG0 generation)
  - Status variables are updated
  - FTM1 modulo and channel 0 compare values are set according to the runtime values (based on the values stored in the FTM0 configuration structure)
- PIT module (represents the delay for software trigger generation from its selection):
  - Clock source = peripheral bus = 60 MHz
  - Timer channel 0 overflow interrupt is enabled
  - Period is set to 40  $\mu$ s (twice the FTM0 period)
  - Channel 0 event is used as an external trigger connected to the FTM0 (as hardware TRIG0)
  - *pit0\_callback* function is called on timer overflow
    - A short pulse is generated on the PTC 16 pin (it represents the FTM0 software trigger generation)
    - Status variables are updated
    - FTM1 modulo and channel 0 compare values are set according to the runtime values (based on the values stored in the FTM0 configuration structure)

#### NOTE

The configuration of the FTM1 and PIT timings is set for the best demonstration of the register update—to separate the moment when the registers are written to from the moment when the trigger occurs.

#### NOTE

Consider the timing inaccuracy of the short pulses generated on the PTC16 pin. These pulses are only a symbolic representation.

## 7.2. Running the example

Compile the project (IAR IDE / Keil IDE) before the first use. For more details on downloading the application (based on the *.out*, *.srec*, or *.bin* files) see *FRDM-K64F Freedom Module User's Guide* (document [FRDMK64FUG](#)).

Use FreeMASTER to demonstrate the registers update (or a debug mode with J-Link connected to change the variables in runtime—using Live Watch window in IAR IDE). Consider using the USB connection between your computer and FRDM-K64F (OpenSDA connector) with a proper OpenSDAv2 application loaded (J-Link CDC UART Port application). Open the FreeMASTER project file *an\_ftm\_buffered\_registers.pmp.pmp* located in the *user\_projects\app\_an\_ftm\_buffered\_registers* folder.

A dedicated FreeMASTER project application opens. Click “Project”, then “Options...”, select the correct number of your virtual serial COM port, and set the baud rate to 115200. Then click “OK” and start the communication (click the red “STOP” icon).

To change the variable according to the flow described below, click the dedicated variable row under the “value” column in the FreeMASTER “Variable Watch” window, and choose from the available values or just type in the corresponding value.

### NOTE

It is considered that FreeMASTER is already installed on your computer.

## 7.3. Code execution flow

The execution of the code is as follows:

- Hardware initialization (using SDK 1.3 with the dedicated board; it includes dedicated pins port clock gate enable, timers clock enable, and system clock configuration)
- Application pins configuration
  - PTC16 and PTC17 are configured as GPIO outputs
  - PTC1 and PTC2 are configured as FTM0 channels’ 0 and 1 outputs
- PIT channel 0, FTM0, and FTM1 initialization according to the previously mentioned configuration
- Infinite loop with the write register status variable checking
- To enable the register update flow, follow these steps:
  - Write the new value to the variable dedicated for a specific register (depends on the register update approach chosen):
    - CNTIN in FreeMASTER (or `ftm0_config.InitVal` in runtime debug)
    - MOD in FreeMASTER (or `ftm0_config.ModuloVal` in runtime debug)
    - C0V in FreeMASTER (or `ftm0_config.ChComVal[HW_CHAN0]` in runtime debug)
    - C1V in FreeMASTER (or `ftm0_config.ChComVal[HW_CHAN1]` in runtime debug)
    - INVCTRL in FreeMASTER (or `ftm0_config.InvCtrl` in runtime debug)
    - OUTMASK in FreeMASTER (or `ftm0_config.OutMask` in runtime debug)
    - SWOCTRL in FreeMASTER (or `ftm0_config.SwOutCtrl` in runtime debug)

### NOTE

Configure the oscilloscope to trigger a single trigger after writing the new value.

- Select the trigger to be used for the registers update by writing *kFtmUseSoftwareTrig* or *kFtmUseHardwareTrig0* to the `ftm0_config.PwmSyncConfig.SyncMethod` dedicated variable
- Write *WREG\_ENABLED* into the *b\_WRegStatus* variable

## 7.4. Selected examples from sample code

This section demonstrates the buffered registers update on examples taken from the sample code. The demonstration is based on signal waveforms' description obtained using an oscilloscope. The oscilloscope channels represent:

- CH1 (yellow)—short pulses
  - The first pulse represents the moment when the register is written to
  - The second pulse represents the moment when the trigger occurs (software or hardware)

### NOTE

Consider that the moment when the pulse is generated is only approximate. It does not reflect the exact moment of register write and trigger approximation.

- CH2 (blue) and CH4 (green)—FTM0 PWM outputs
- CH3 (purple)—FTM0 counter overflow (boundary cycles)

Most of the examples provided below represent the channel compare value update (especially the C0V and C1V) to demonstrate the pulse width change. The C0V values are changed from -1500 to -2500 and the C1V values are changed from 1500 to 2500.

### NOTE

Consider that the TPM approach uses the independent mode for both channels (CH1 inverted to CH0). It means that the two channels generate two independent pulse widths. The other two approaches use the combine mode with complementary pairs. It means that the C0V value represents the rising edge and the C1V value represents the falling edge of the PWM output. The CH0 and CH1 outputs are complementary to each other (no dead-time is implemented).

### 7.4.1. TPM approach

Choose the TPM approach for the buffered register update using this definition:

```
#define PWM_SYNCHRONIZATION_METHOD    TPM_APPROACH
```

The following figure shows the time waveforms of FTM0 module's C0V and C1V registers update. The C0V and C1V registers are loaded with their new values in the moment represented by the short pulse on channel 1 (yellow). The register update occurs when the counter changes from the MOD value to the CNTIN value, as shown in the following figure on the next rising/falling edge of channel 3 (purple).

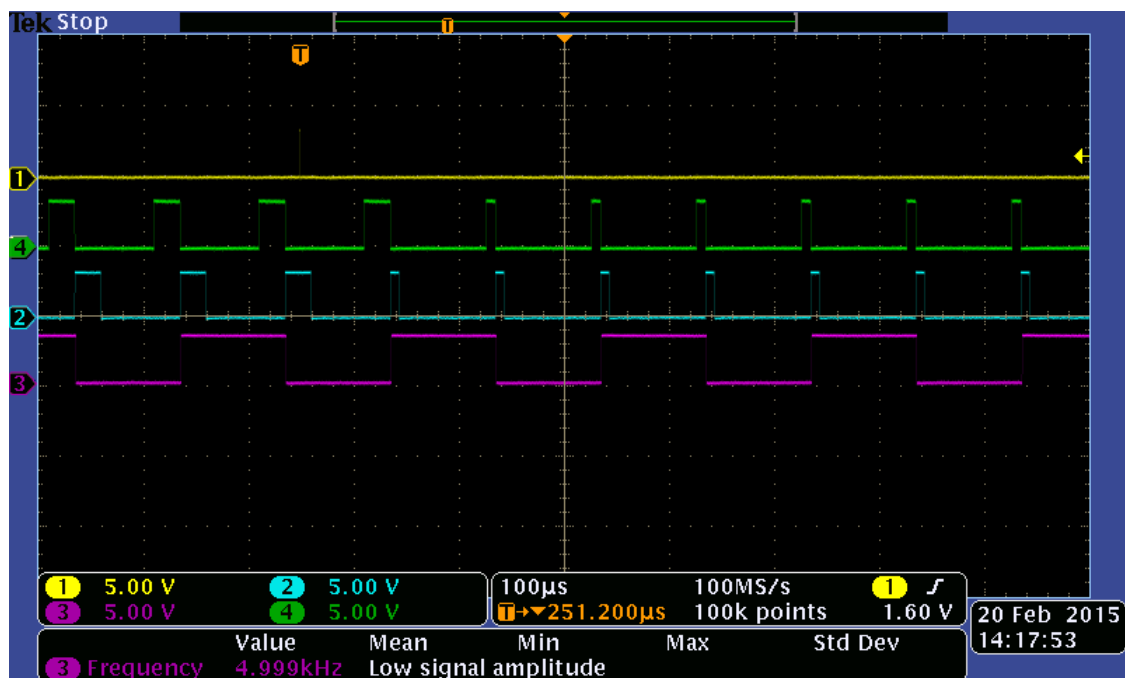


Figure 35. TPM approach—CnV register update

## 7.4.2. Legacy PWM synchronization

The legacy PWM synchronization for the buffered register update is chosen using this definition:

```
#define PWM_SYNCHRONIZATION_METHOD          LEGACY_PWM_SYNCHRONIZATION
```

Figure 36 and Figure 37 show the time waveforms of the C0V and C1V registers update by the legacy PWM synchronization using software and hardware triggers. The C0V and C1V registers are loaded with their new values in the moment represented by the first short pulse on channel 1 (yellow). The second pulse width represents the trigger generation. The register update occurs at the loading point after the trigger occurs, that means on the next rising/falling edge on channel 3 (purple).

Figure 38 and Figure 39 represent the same scenario with counter re-initialization enabled. The C0V and C1V registers are loaded with their new values in the moment represented by the first short pulse on channel 1 (yellow). The second pulse width represents the trigger generation. The register update occurs one system clock cycle after the trigger generation due to counter re-initialization. This is shown in Figure 38 and Figure 39 by the elongated FTM0 period (unfinished period + one full period) on channel 3 (purple).

### NOTE

The legacy PWM synchronization example is shown and included in the sample code only to demonstrate that it is a part of the FTM module. However, it is recommended to use the enhanced PWM synchronization instead of the legacy PWM synchronization because the enhanced PWM synchronization already includes all features of the legacy PWM synchronization.

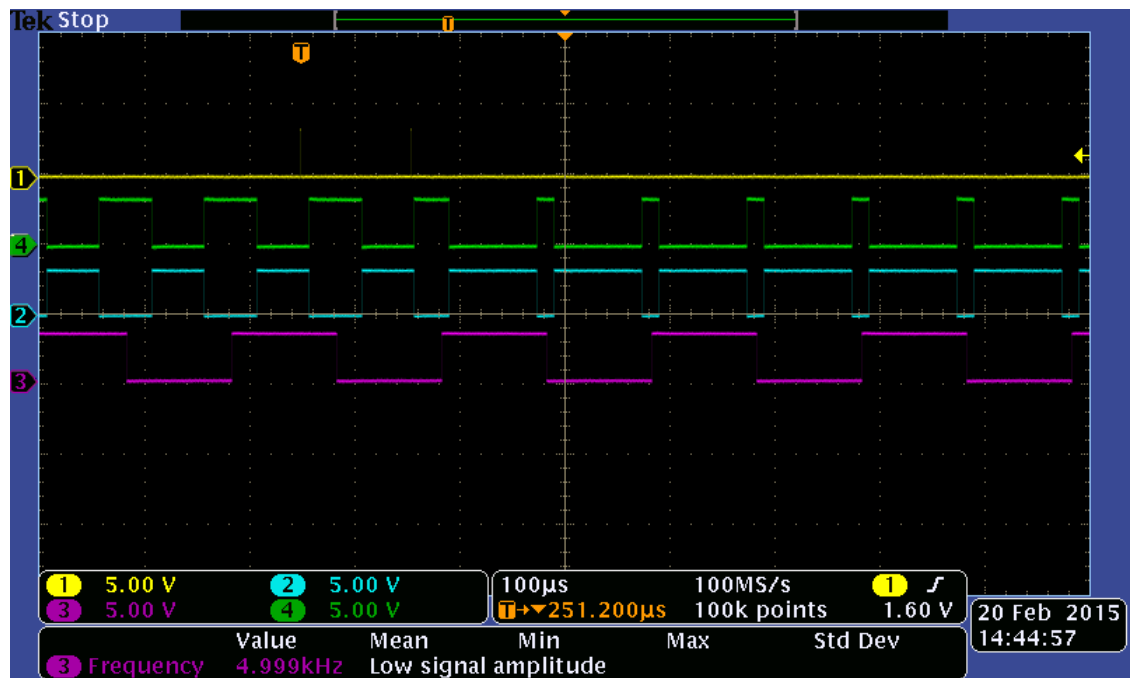


Figure 36. Legacy PWM synchronization—CnV register update on software trigger

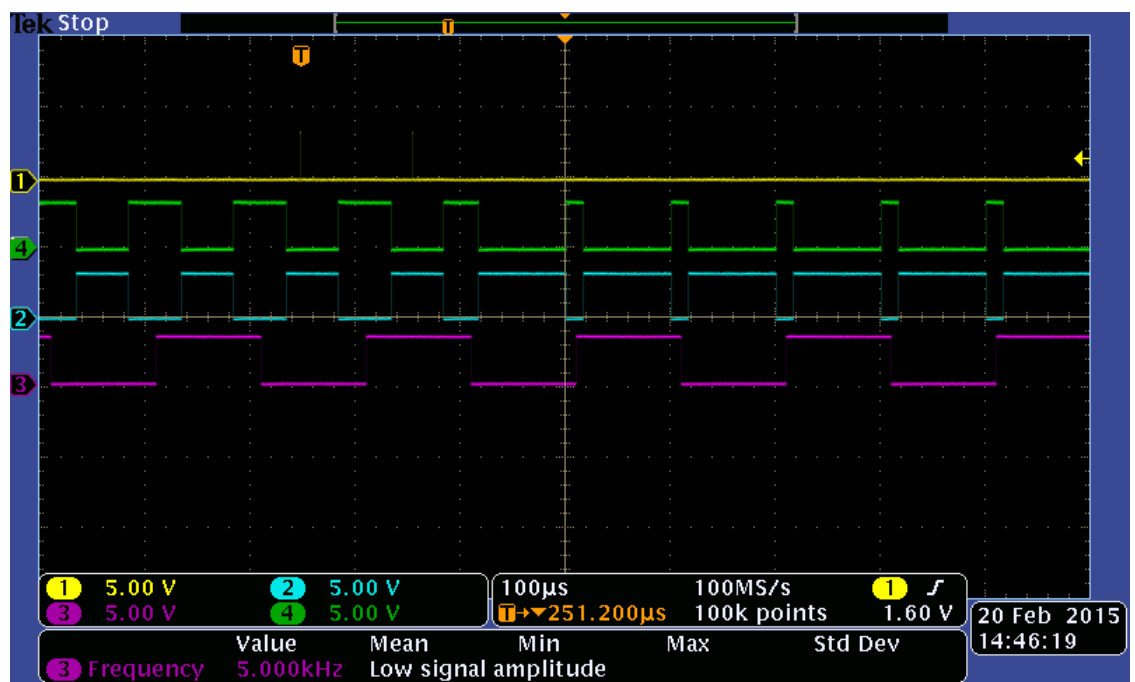


Figure 37. Legacy PWM synchronization—CnV register update on hardware trigger

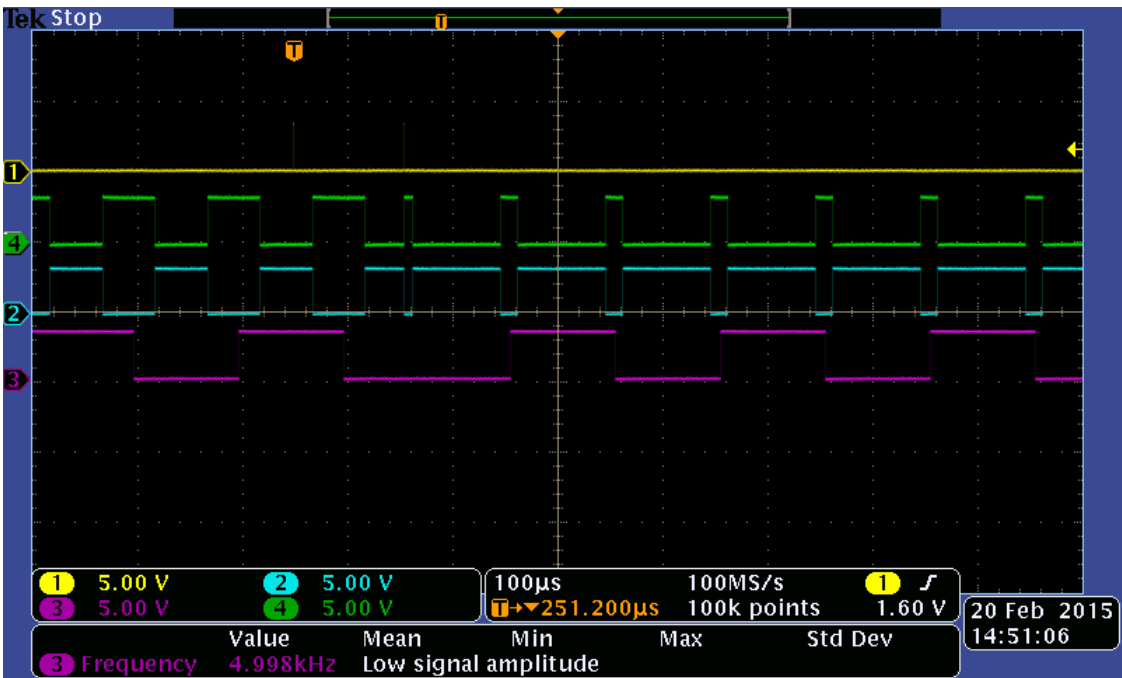


Figure 38. Legacy PWM synchronization—CnV register update with counter re-initialization on software trigger

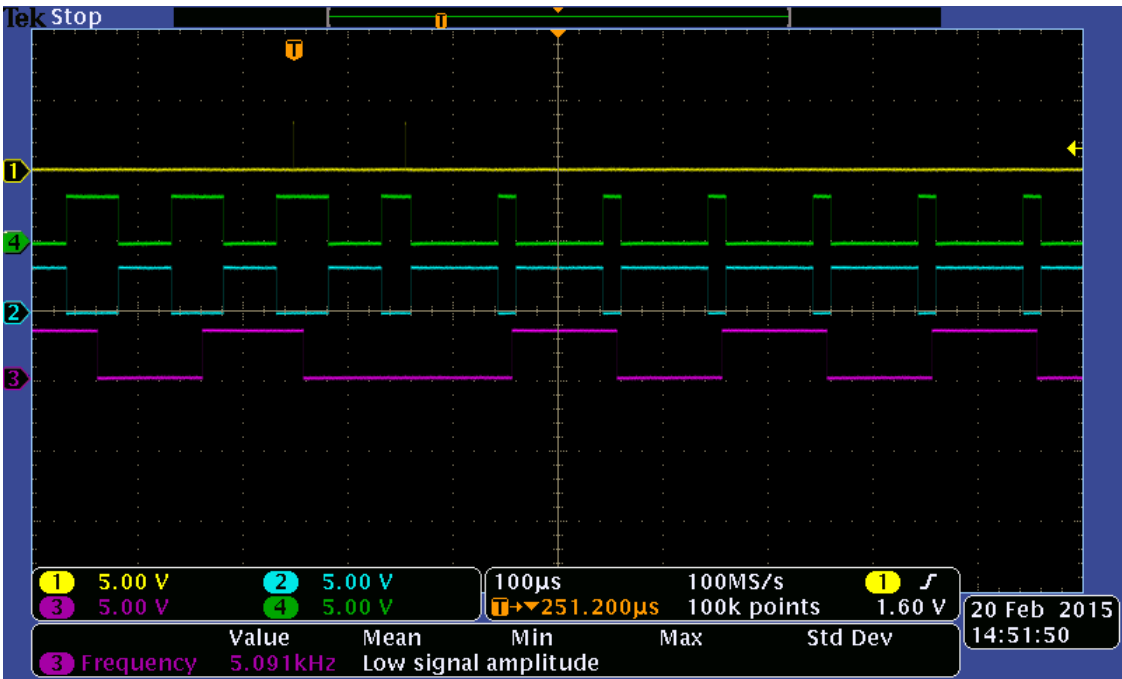


Figure 39. Legacy PWM synchronization—CnV register update with counter re-initialization on hardware trigger



### 7.4.3. Enhanced PWM synchronization

Choose the enhanced PWM synchronization for the buffered register update using this definition:

```
#define PWM_SYNCHRONIZATION_METHOD      ENHANCED_PWM_SYNCHRONIZATION
```

Figure 40 and Figure 41 show the time waveforms of the C0V and C1V registers updated by the enhanced PWM synchronization using the software and hardware triggers. The C0V and C1V registers are loaded with their new values in the moment represented by the first short pulse on channel 1 (yellow). The second pulse width represents the trigger generation. The register update occurs at the loading point after the trigger occurs, that means on the next rising/falling edge of channel 3 (purple).

Figure 42 and Figure 43 represent the same scenario with the counter reset on trigger enabled (similar to the legacy PWM synchronization with re-initialization enabled). The C0V and C1V registers are loaded with their new values in the moment represented by the first short pulse on channel 1 (yellow). The second pulse width represents the trigger generation. The register update occurs one system clock cycle after the trigger generation due to counter reset. This is shown in Figure 38 and Figure 39 by the elongated FTM0 period (unfinished period + one full period) on channel 3 (purple).

Figure 44 and Figure 45 represent the INVCTRL register update. The dedicated channel pair bit in the INVCTRL register is loaded with the new value in the moment represented by the first short pulse on channel 1 (yellow). The second pulse width represents the trigger generation. The register update occurs one system clock cycle after the trigger generation. This is shown in Figure 44 and Figure 45, channels 2 and 4 (blue and green), by the inverted PWM outputs.

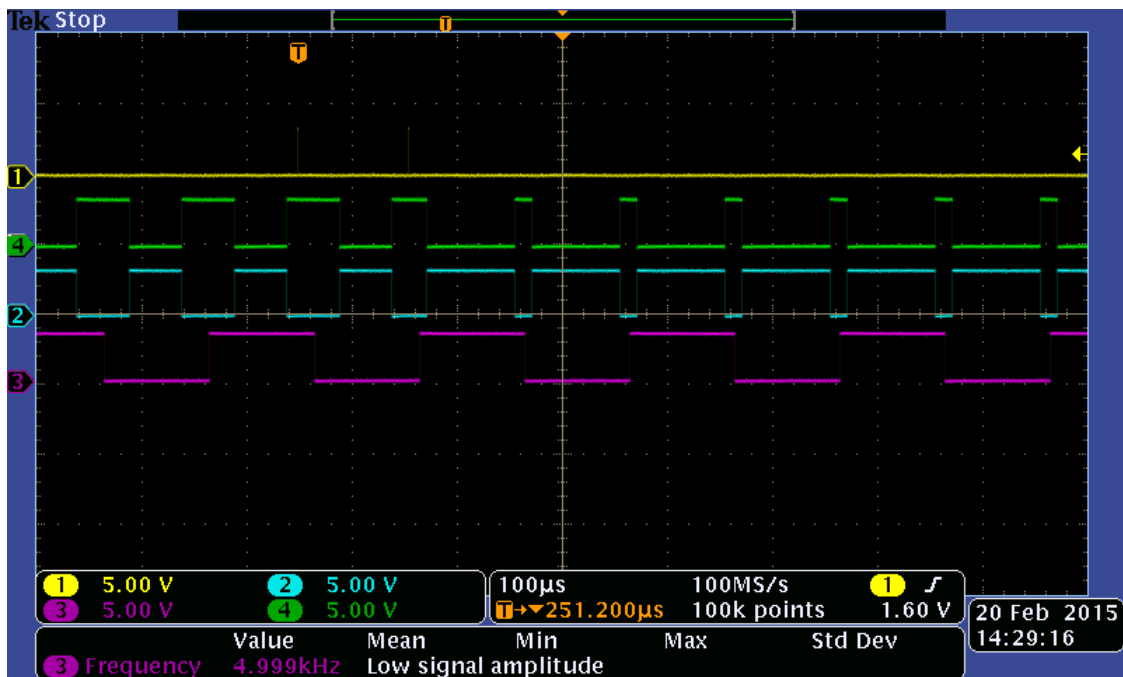


Figure 40. Enhanced PWM synchronization—CnV register update on software trigger

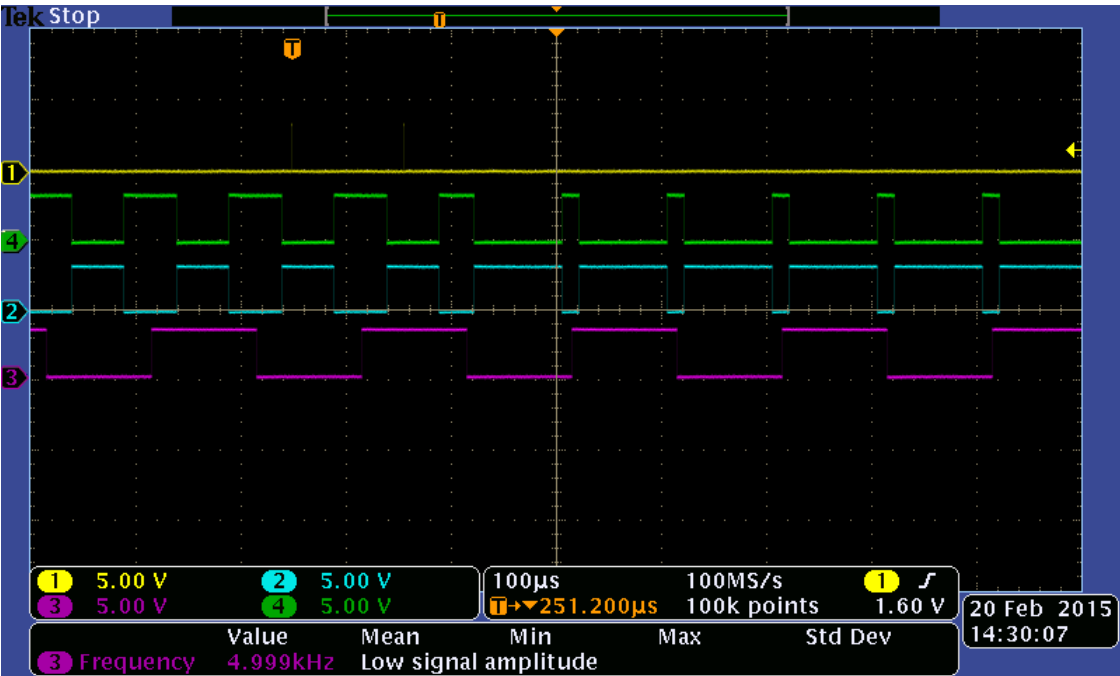


Figure 41. Enhanced PWM synchronization—CnV register update on hardware trigger

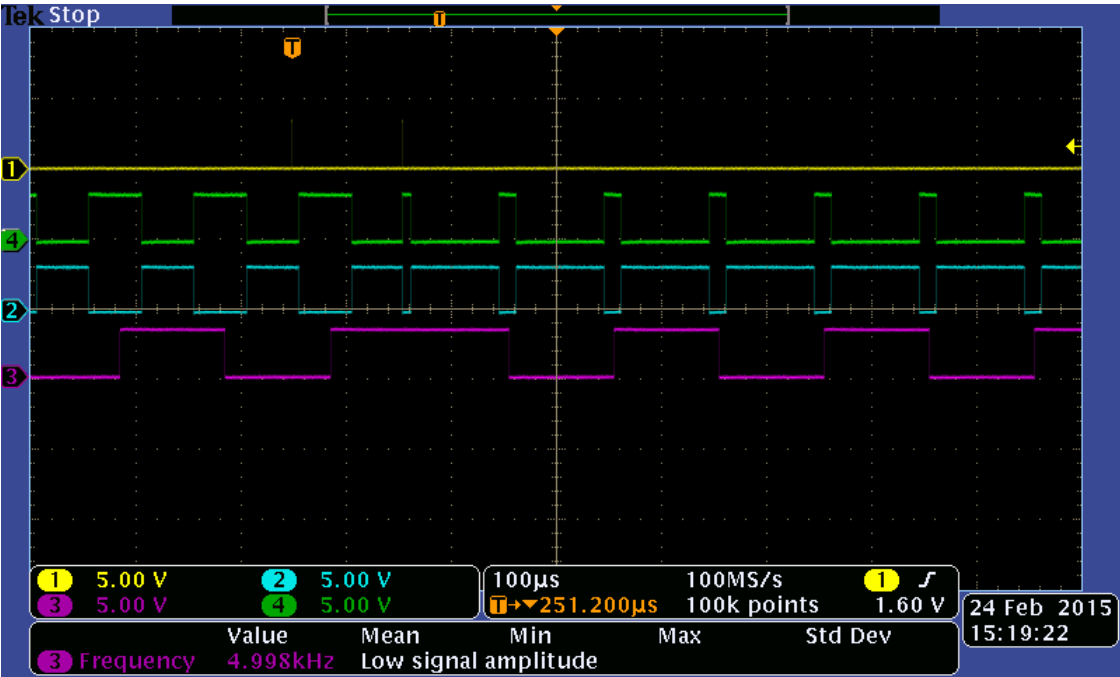


Figure 42. Enhanced PWM synchronization—CnV register update with counter reset on software trigger

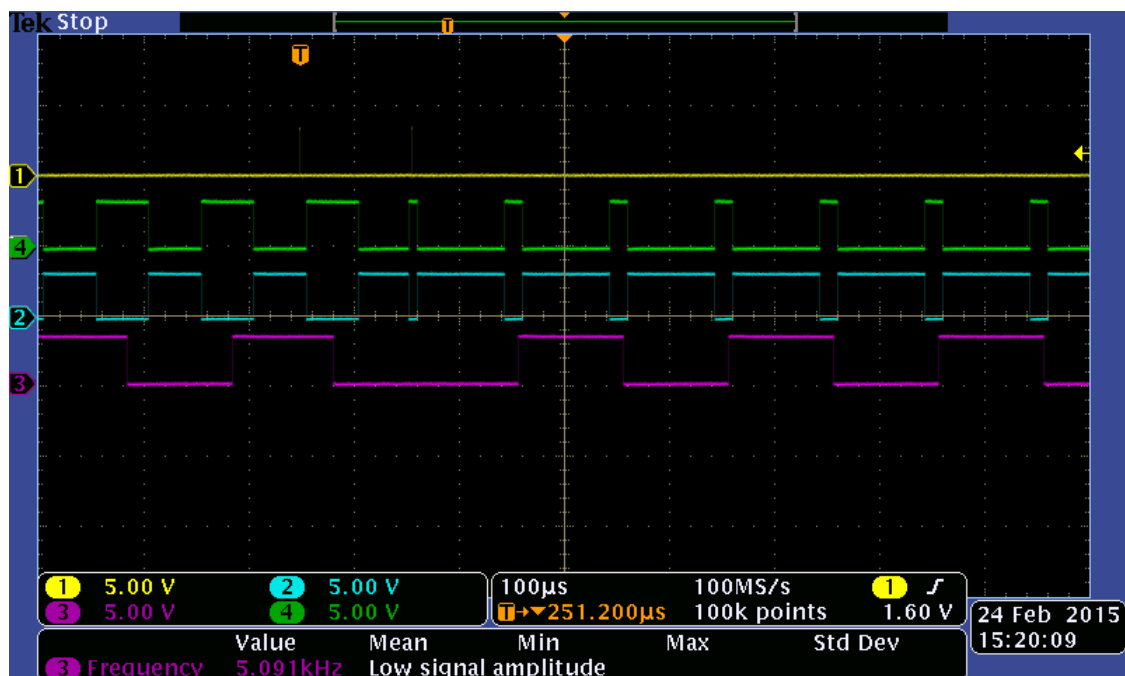


Figure 43. Enhanced PWM synchronization—CnV register update with counter reset on hardware trigger

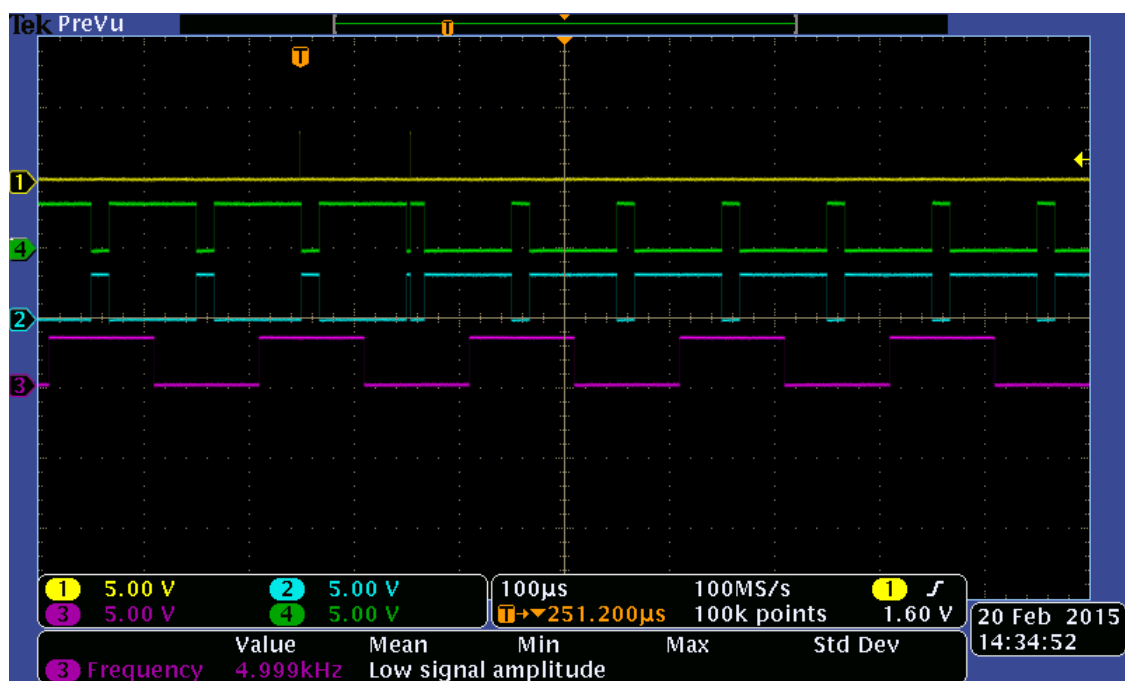


Figure 44. Enhanced PWM synchronization—INVCTRL register update on software trigger

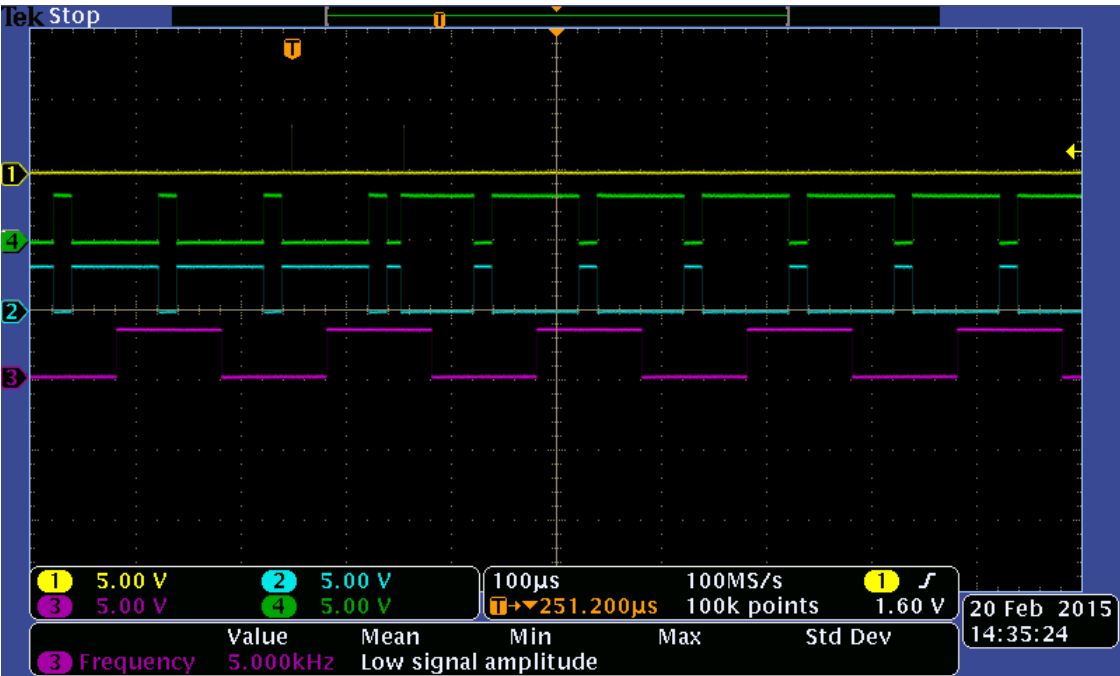


Figure 45. Enhanced PWM synchronization—INVCTRL register update on hardware trigger

8. References

- *K64 Sub-Family Data Data Sheet* (document [K64P144M120SF5](#))
- *K64 Sub-Family Reference Manual* (document [K64P144M120SF5RM](#))

9. Revision History

This table summarizes the changes made to this document since the initial release:

Table 1. Revision history

Revision number	Date	Substantive changes
0	03/2016	Initial release.

---

**How to Reach Us:**

**Home Page:**  
[freescale.com](http://freescale.com)

**Web Support:**  
[freescale.com/support](http://freescale.com/support)

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions).

Freescale, the Freescale logo, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners. ARM, the ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 Freescale Semiconductor, Inc.

Document Number: AN5261  
Rev. 0  
03/2016

