AN14024

LPC86x FlexTimer Module Feature

Rev. 1.0 — 29 September 2023 **Application note**

Document information

Information	Content
Keywords	AN14024, FlexTimer, LPC86x, PWM, motor control, power energy
Abstract	This document introduces the FlexTimer module feature, which is an enhanced version of CTimer used for motor control and power energy applications.

1 Introduction

The FlexTimer module (FTM) is an enhanced timer module when compared with the Timer/PWM module (TPM). The FTM is commonly used in the Kinetis series MCUs and for motor control, lighting, and power-conversion application. This document introduces the features of the FTM module in LPC86x.

The key features of the FTM module are as follows:

- Each channel can be configured for Input capture, Output compare, or Edge-aligned or Center-aligned PWM mode.
- Each pair of channels can be combined to generate PWM signals with equal outputs, pairs with complementary outputs, or independent outputs.
- The dead time insertion is available for each complementary pair.
- Quadrature decoder with input filters, relative position counting, and interrupt on position count or capture of position count on external event.
- Software control of PWM outputs, fault inputs for global fault control.

The masking, inverting, polarity and fault control, and hardware dead time insertion are the main features of the FTM module dedicated for motor-control applications. They provide greater flexibility and significantly reduce the CPU load. If the FTM module is not used for motor control, it retains standard timer functions such as the Input capture or Output compare modes.

2 Acronyms

Table 1 lists the acronyms used in this document.

Table 1. Acronyms

Acronym	Meaning
FTM	FlexTimer module
EPWM	Edge-aligned PWM
CPWM	Center-aligned PWM
DCAP	Dual-edge capture
GTB	Global time base

3 FlexTimer overview

The LPC86x FTM is a six-to-four-channel timer that supports the following features:

- Input capture
- Output compare
- Generation of PWM signals to control electric motor
- Power management applications

The FTM time reference is a 16-bit counter that can be used as an unsigned or signed counter.

The LPC86x device has two FlexTimers as follows:

- *FTM0*: FTM0 provides six channels and includes support for motor control including fault control.
- *FTM1*: FTM1 provides four channels. This timer does not include fault control but includes a quadrature decoder.

Both the FlexTimers can be clocked up to 60 MHz. When the FlexTimers are operated at a rate higher than the CPU, they must be exactly two times the CPU/AHB frequency. Otherwise, they can use the same clock as the CPU/AHB. Figure 1 shows the FTM block diagram.

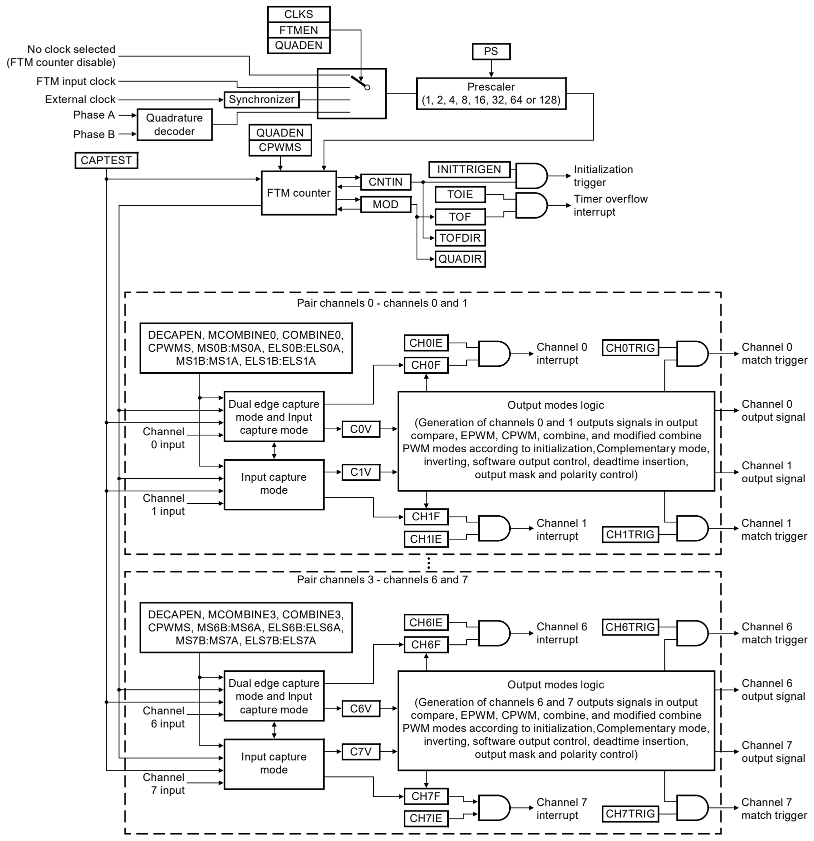

CLKS
FTMEN
QUADEN
PS
No clock selected (FTM counter disable)
FTM input clock
External clock
Synchronizer
Prescaler (1, 2, 4, 8, 16, 32, 64 or 128)
Phase A
Phase B
Quadrature decoder
QUADEN
CPWMS
CAPTEST
FTM counter
CNTIN
MOD
INITTRIGEN
Initialization trigger
TOIE
TOF
Timer overflow interrupt
TOFDIR
QUADIR
Pair channels 0 - channels 0 and 1
DECAPEN, MCOMBINE0, COMBINE0, CPWMS, MS0B:MS0A, ELS0B:ELS0A, MS1B:MS1A, ELS1B:ELS1A
CH0IE
CH0F
Channel 0 interrupt
CH0TRIG
Channel 0 match trigger
Dual edge capture mode and Input capture mode
Channel 0 input
C0V
Output modes logic
(Generation of channels 0 and 1 outputs signals in output compare, EPWM, CPWM, combine, and modified combine PWM modes according to initialization, Complementary mode, inverting, software output control, deadtime insertion, output mask and polarity control)
Channel 0 output signal
Channel 1 output signal
Input capture mode
Channel 1 input
C1V
CH1F
CH1IE
Channel 1 interrupt
CH1TRIG
Channel 1 match trigger
Pair channels 3 - channels 6 and 7
DECAPEN, MCOMBINE3, COMBINE3, CPWMS, MS6B:MS6A, ELS6B:ELS6A, MS7B:MS7A, ELS7B:ELS7A
CH6IE
CH6F
Channel 6 interrupt
CH6TRIG
Channel 6 match trigger
Dual edge capture mode and Input capture mode
Channel 6 input
C6V
Output modes logic
(Generation of channels 6 and 7 outputs signals in output compare, EPWM, CPWM, combine, and modified combine PWM modes according to initialization, Complementary mode, inverting, software output control, deadtime insertion, output mask and polarity control)
Channel 6 output signal
Channel 7 output signal
Input capture mode
Channel 7 input
C7V
CH7F
CH7IE
Channel 7 interrupt
CH7TRIG
Channel 7 match trigger

Figure 1. FTM block diagram

Figure 2 shows the FTM module channel modes setting and the capture edge level.

DECAPEN	COMBINE	CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
0	0	0	0	1	Input capture	Capture on rising edge only
				10		Capture on falling edge only
				11		Capture on rising or falling edge
			1	1	Output compare	Toggle output on match
				10		Clear output on match
				11		Set output on match
			1X	10	Edge-aligned PWM	High-true pulses (clear output on match)
				X1		Low-true pulses (set output on match)
		1	XX	10	Center-aligned PWM	High-true pulses (clear output on match-up)
				X1		Low-true pulses (set output on match-up)
	1	0	XX	10	Combine PWM	High-true pulses (set on channel (n) match, and clear on channel (n+1) match)
				X1		Low-true pulses (clear on channel (n) match, and set on channel (n+1) match)
1	0	0	X0		Dual-edge capture mode	One-shot capture mode
			X1			Continuous capture mode

Figure 2. FTM channel mode setting

4 FlexTimer features

The FTM features are discussed in detail in further sections.

4.1 Edge-aligned PWM mode

For Edge-aligned PWM (EPWM) mode, the FTM counter counts from the `FTM_CNTIN` value to the `FTM_MOD` value. All FTM channel signals align at the edge when the FTM counter changes from the MOD value to the CNTIN value.

The Edge-aligned mode is selected when:

- QUADEN = 0
- DECAPEN = 0
- COMBINE = 0
- CPWMS = 0
- MSnB = 1

The edge-aligned PWM period can be determined from Equation 1:

$$(\mathrm{MOD} - \mathrm{CNTIN} + 0\,\mathrm{x}\,0001) \tag{1}$$

The pulse width or the duty cycle can be determined from Equation 2 or Equation 3, depending on the ELSnB:ELSnA bits setting.

$$(CnV - CNTIN) \tag{2}$$

$$(MOD - NTIN - CnV) \tag{3}$$

The Edge-aligned mode PWM code example is as follows:

```
void FTM_EdgeAlignedMode_Output(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo in initialization stage (10kHz PWM frequency @60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN in initialization stage */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* Enable high-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
/* Set channel value in initialization stage */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
/* Reset FTM counter */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
}
```

Figure 3 shows that the CH1 and CH2 channel signals on the oscilloscope are the `FTM0_CH0` and `FTM0_CH1` signals. The `FTM0_CH0` and `FTM0_CH1` are aligned in the raising edge.

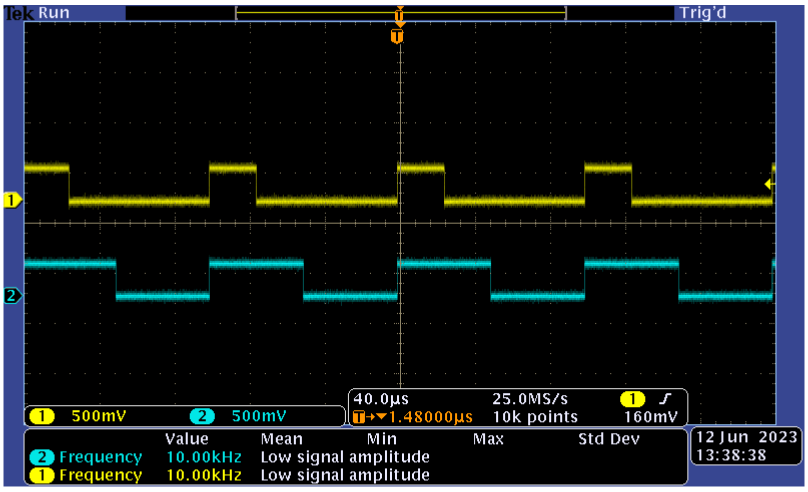


Figure 3. Edge-aligned PWM signal

4.2 Center-aligned PWM mode

In Center-aligned PWM (CPWM) mode, the FTM counter counts up from `FTM_CNTIN` to `FTM_MOD` and then counts down from `FTM_MOD` to `FTM_CTNIN`. All FTM channel signals align at the point when the FTM counter reaches up to `FTM_MOD` value.

The Center-aligned mode is selected when:

- QUADEN = 0
- DECAPEN = 0
- COMBINE= 0
- CPWMS = 1

The center-aligned PWM period can be determined from Equation 4:

$$2 \times (MOD - CNTIN + 0 \times 0001) \tag{4}$$

The pulse width or the duty cycle can be determined from Equation 5 or Equation 6, depending on the ELSnB:ELSnA bits setting.

$$2 \times (CnV - CNTIN) \tag{5}$$

$$2 \times (MOD - CNTIN - CnV) \tag{6}$$

The Center-aligned mode PWM code example is as follows:

```
void FTM_CenterAlignedMode_Output(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1);
/* Set CNTIN in initialization stage */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1500); // 50% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(750); // 25% duty cycle
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
}
```

Figure 4 shows that the CH1 and CH2 channel signals on the oscilloscope are the `FTM0_CH0` and `FTM0_CH1` signals. The `FTM0_CH0` and `FTM0_CH1` are aligned in the center.

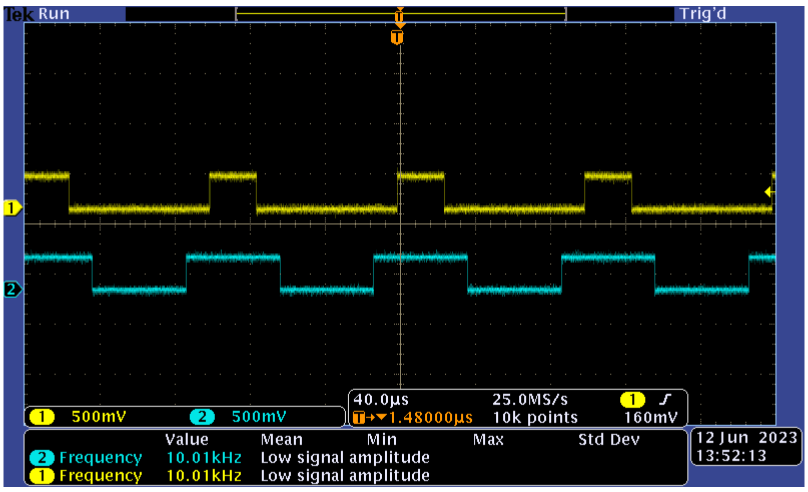


Figure 4. Center-aligned PWM signal

4.3 Complementary mode and dead time insertion

The FTM module supports the Complementary mode. If the COMP bit enables the Complementary mode in the `FTM_COMBINE` register, the even FTM channel generates the output signal. The complementary logic generates the odd output signal as a complement to the even FTM channel. The complementary signal generation can be set individually for each pair of the FTM outputs.

To avoid short-circuit, the dead time must be inserted into the complementary signals. The dead time insertion is provided by the dead time logic, following the complementary logic. The DTEN bit enables this feature in the `FTM_COMBINE` register. The dead time logic delays every rising edge by a time set in the `FTM_DEADTIME` register.

The dead time consists of two parts as follows:

- The first two most significant bits `DTPS[1:0]` define the pre-scaler of the system clock.
- The bits `DTVAL[5:0]` define the dead time value using the pre-scaled clock.

The Complementary mode and dead time insertion are applied to both the Edge-aligned PWM and Center-aligned PWM modes, described in Section 4.1 and Section 4.2.

The Complementary mode and dead time insertion PWM code example is as follows:

```
void FTM_CompMode_Output(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable combine, complementary mode and dead-time for channel pair CH0/CH1*/
FTM0->COMBINE = FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK
| FTM_COMBINE_COMP1_MASK | FTM_COMBINE_DTEN1_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Select up-down counter for Center-Align PWM */
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[2].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[3].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(1500); // 25% duty cycle
FTM0->CONTROLS[3].CnV=FTM_CnV_VAL(1500); // 25% duty cycle
FTM0->DEADTIME = FTM_DEADTIME_DTVAL(10) | FTM_DEADTIME_DTPS(1);
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK
| FTM_SC_PWMEN2_MASK | FTM_SC_PWMEN3_MASK;
}
```

Figure 5 shows the following parameters:

- CH1 and CH2 in the oscilloscope are `FTM0_CH0` and `FTM0_CH1`.
- CH3 and CH4 are `FTM0_CH2` and `FTM0_CH3`.
- The `FTM0_CH0` and `FTM0_CH1` and `FTM0_CH2` and `FTM_CH3` are all in Complementary mode.
- The `FTM0_CH0` and `FTM0_CH1` PWM duty cycles are 50 %.
- The `FTM0_CH2` and `FTM0_CH3` PWM duty cycles are 25 %.

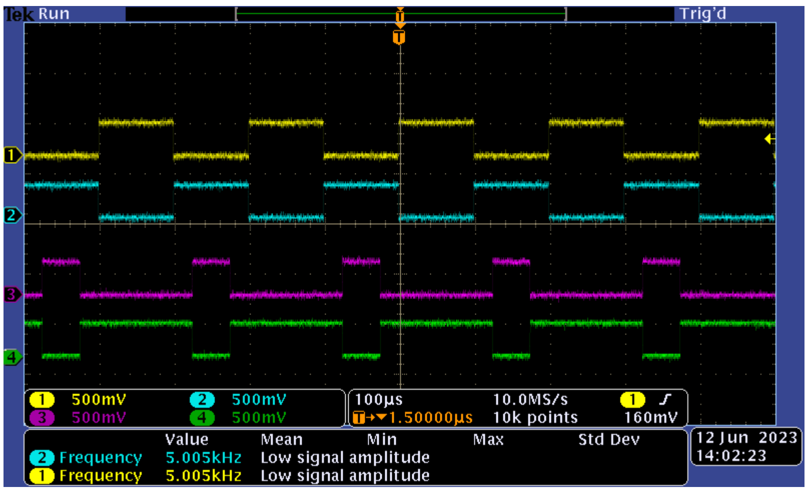


Figure 5. FTM Complementary mode PWM output

Figure 6 shows the dead time value between `FTM0_CH0` and `FTM0_CH1`. The oscilloscope can get the dead time value of 0.16 µs. This value is equal to the value, which has been set in the code example.

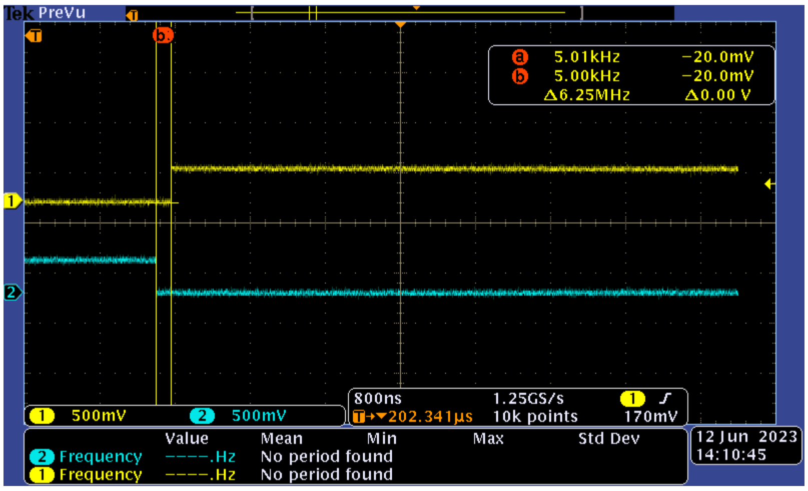


Figure 6. FTM dead time insertion

4.4 Combine mode

The Combine mode provides a higher flexibility because the PWM channel (n) output is generated by combining the even channel (n) and the adjacent odd channel (n+1). This implies that the even and odd channels must work in the Complementary mode.

The Combine mode enables generating the EPWM and CPWM using only the up counter, the asymmetrical PWM, or the phase-shifted PWM. The phase-shifted PWM generation is commonly used in phase-shifted full-bridge converters and motor-control applications. Here, the 3-phase stator currents are reconstructed from the current sensed by a single shunt resistor placed in the DC-link and the actual combination of the power supply inverter switches.

The Combine mode is selected when:

- QUADEN = 0
- DECAPEN = 0
- MCOMBINE = 0
- COMBINE = 1
- CPWMS = 0

To generate a phase-shifted PWM with high-true pulses, set the control bits as ELSnB:ELSnA = 1:0. This code example shows the configuration of the FTM0 module used for the phase-shifted PWM generation.

Complementary mode and phase shift PWM code example is as follows:

```
void FTM_CombineMode_Output(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Enable combine, complementary mode and dead-time for channel pair CH0/CH1*/
FTM0->COMBINE = FTM_COMBINE_COMBINE0_MASK | FTM_COMBINE_DTEN0_MASK
| FTM_COMBINE_COMBINE1_MASK | FTM_COMBINE_DTEN1_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[2].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[3].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1000); // 50% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(4000); // 50% duty cycle
FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(2500); // 50% duty cycle
FTM0->CONTROLS[3].CnV=FTM_CnV_VAL(5500); // 50% duty cycle
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK
| FTM_SC_PWMEN2_MASK | FTM_SC_PWMEN3_MASK;;
}
```

Figure 7 shows the following parameters:

- The scope channels CH1, CH2, CH3, and CH4 represent the FTM0 module channels `FTM0_CH0`, `FTM0_CH1`, `FTM0_CH2`, and `FTM0_CH3`, respectively.
- The first channel pair (`FTM0_CH0`/`FTM0_CH1`) and the second channel pair (`FTM0_CH2`/`FTM0_CH3`) both work in the Complementary mode with a 50 % duty cycle.
- The second channel pair (`FTM0_CH2`/`FTM0_CH3`) is phase-shifted by 90 degrees to the first channel pair (`FTM0_CH0`/`FTM0_CH1`).

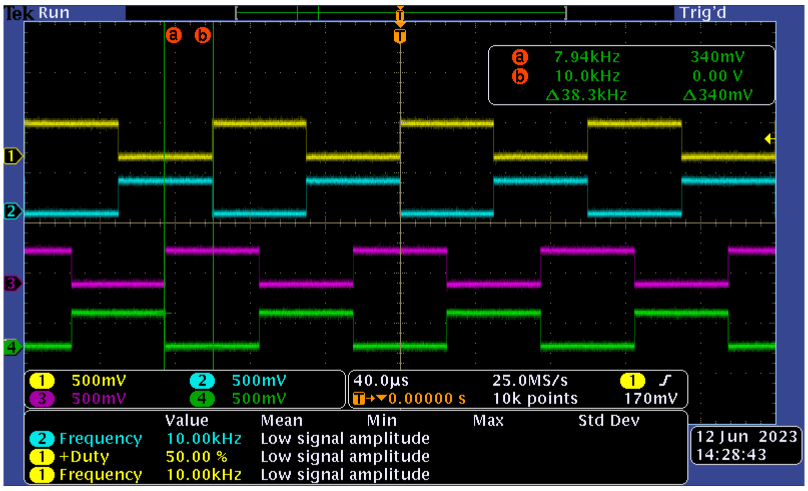


Figure 7. FTM phase-shift PWM output

4.5 Single-edge capture mode

The FTM capture mode has the following uses:

- The FTM capture mode determines the pulse width or the period of the tested signal.
- The FTM capture mode detects the rising/falling edge of an external signal and generates an interrupt to notify that an external event has appeared.
- The FTM capture mode is used in BLDC motor-control applications. The hall sensors in these applications are used to detect the position of the rotor and compute the rotor speed, so that the speed loop can be established. The hall sensors are connected to the channels of the independent FTM (`FTM_CHx`). The FTM can then detect both the falling and rising edges of the hall sensor signals and generates a capture interrupt. In the capture-interrupt routine, the duty cycles of the PWM signals are then modified according to the hall sensor logic.

The Single-edge capture mode is selected when:

- DECAPEN = 0
- MCOMBINE = 0
- COMBINE = 0
- CPWMS = 0
- MSnB:MSnA = 0:0

- ELSnB:ELSnA ≠ 0:0

To measure either the pulse width or the period of the tested signal, perform the following steps:

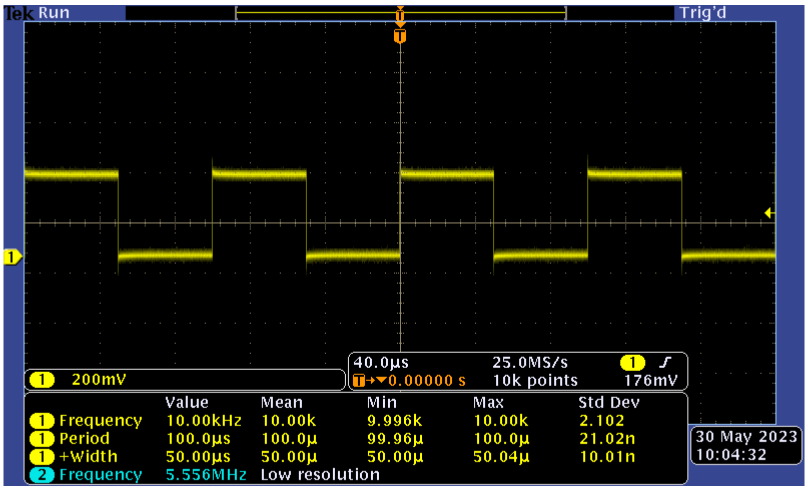
1. Select the input channel of the FTM module `FTM_CHx` and the edge-sensitive input by the control bits ELSnB:ELSnA.
2. When the selected edge occurs on the channel input, the current value of the FTM counter is captured in the CnV register. It also generates a channel interrupt (if CH(n)IE = 1).
3. In the interrupt routine, save the value of the CnV register into a variable.
4. Create a difference between the current value and the saved value from the previous interrupt routine:
   - If the selected capture mode is sensitive either on the rising edge (ELSnB:ELSnA= 0:1) or the falling edge (ELSnB:ELSnA= 1:0), the difference is equal to the signal period.
   - If the selected capture mode is sensitive on both edges (ELSnB:ELSnA = 1:1), the difference is equal to the pulse width of the tested signal.

The Single-edge capture mode code example is as follows:

```
void FTM0_IRQHandler(void)
{
if ((FTM_GetStatusFlags(FTM0) & kFTM_Chnl0Flag) == kFTM_Chnl0Flag)
{
/* Clear interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_Chnl0Flag);
}
ftmIsrFlag = true;
g_index++;
__DSB();
}
```

```
void FTM_SingleEdgeInputCaptureMode(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Input capture mode sensitive on rising edge to measure period of tested signal */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSA_MASK | FTM_CnSC_CHIE_MASK;
/* Reset counter */
FTM0->CNT = 0;
/* Select clock */
FTM0->SC = FTM_SC_CLKS(1);
}
```

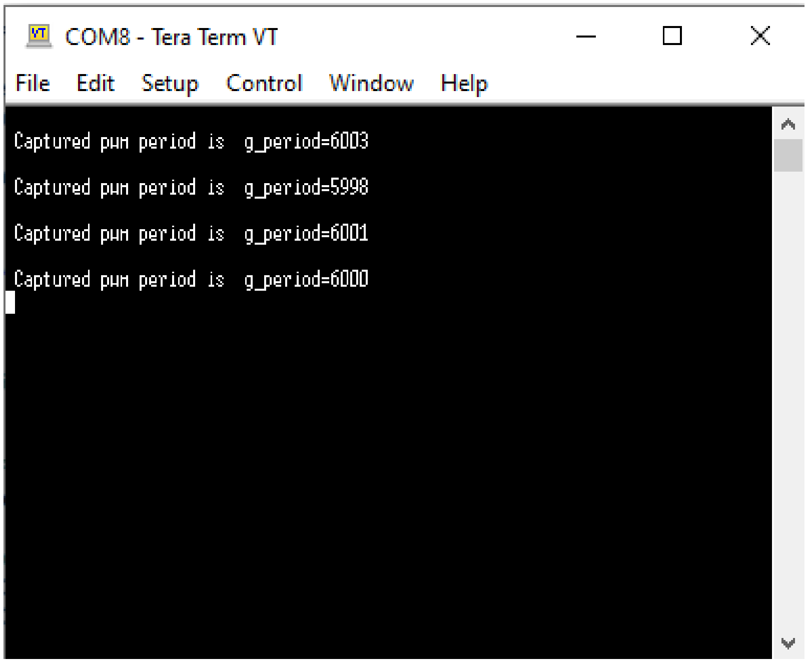
Figure 8 shows the PWM signal attached to the FTM0 used for single edge capture. The PWM signal frequency is 10 kHz.



```
            Value       Mean      Min       Max       Std Dev
1 Frequency 10.00kHz    10.00k    9.996k    10.00k    2.102
1 Period    100.0µs     100.0µ    99.96µ    100.0µ    21.02n
1 +Width    50.00µs     50.00µ    50.00µ    50.04µ    10.01n
2 Frequency 5.556MHz    Low resolution
```

Figure 8. FTM single capture PWM input signal

Figure 9 shows the calculated input PWM signal period MOD value. This value also determines its PWM period.



```
Captured pwm period is  g_period=6003
Captured pwm period is  g_period=5998
Captured pwm period is  g_period=6001
Captured pwm period is  g_period=6000
```

Figure 9. FTM single capture PWM period value

4.6 Dual-edge capture mode

The Dual-edge capture mode uses two FTM channels that enable measuring the positive-polarity or negative-polarity pulse width of the signals. In this mode, input the signals through the even FTM channels and ignore the odd channels.

The Dual-edge capture mode is selected when DECAPEN = 1. The Dual-edge capture mode of the FTM can work either in the One-shot capture mode or the Continuous capture mode. The One-shot capture mode is selected when MS(n)A = 0. If the DECAP bit is enabled, the edges are captured. For every new measurement, clear the CH(n)F and CH(n+1)F and set the DECAP bit again. The Continuous capture mode is selected when MS(n)A = 1. In this mode, if the DECAP bit is set, the edges are captured continuously. For each new measurement, it is necessary to clear the CH(n)F and CH(n+1)F bits.

To measure the positive-polarity pulse width of the tested signal (either in the One-shot mode or in the Continuous mode), configure the channels as follows:

- To capture the rising edge (ELS(n)B:ELS(n)A = 1:0), configure channel (n).
- To capture the falling edge (ELS(n+1)B:ELS(n+1)A = 0:1), configure channel (n+1).

When a second falling edge of the tested signal is detected, set CH(n+1)F, clear the DECAP bit, and if CH(n+1)IE=1, generate an interrupt. In the interrupt routine, subtract the values saved in the C(n+1)V and C(n)V registers. The subtraction determines the positive-polarity pulse width of the tested signal and clears the CH(n+1)F bit.

If the application requires to measure the negative-polarity pulse width of the tested signal, configure the channels as follows:

- To capture the falling edge (ELS(n)B:ELS(n)A = 0:1), configure channel (n).
- To capture the rising edge (ELS(n+1)B:ELS(n+1)A = 1:0), configure channel (n+1).

To determine the period of the tested signal, channel (n) and channel (n+1) must be sensitive on the same edges.

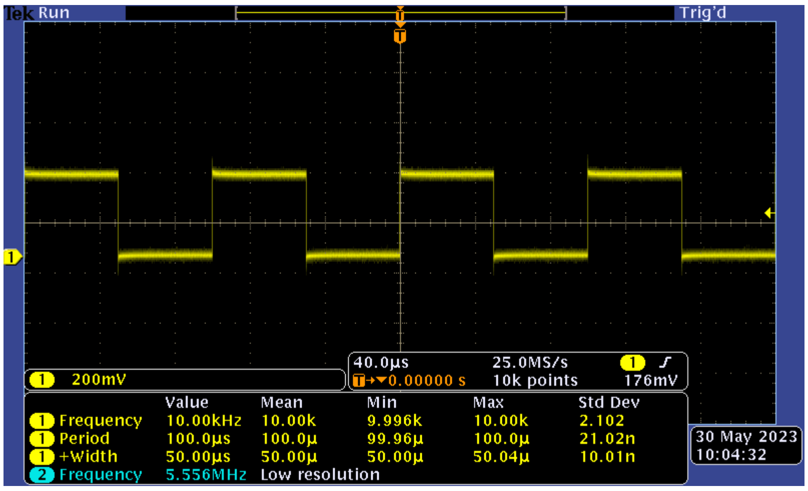
Dual-edge capture mode code example is as follows:

```
void FTM0_IRQHandler(void)
{
if ((FTM_GetStatusFlags(FTM0) & kFTM_TimeOverflowFlag) == kFTM_TimeOverflowFlag)
{
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_TimeOverflowFlag);
g_timerOverflowInterruptCount++;
}
else if (((FTM_GetStatusFlags(FTM0) & kFTM_Chnl0Flag) == kFTM_Chnl0Flag) &&
 (ftmFirstChannelInterruptFlag == false))
{
/* Disable first channel interrupt.*/
FTM_DisableInterrupts(FTM0, kFTM_Chnl0InterruptEnable);
g_firstChannelOverflowCount = g_timerOverflowInterruptCount;
ftmFirstChannelInterruptFlag = true;
}
else if ((FTM_GetStatusFlags(FTM0) & kFTM_Chnl1Flag) == kFTM_Chnl1Flag)
{
/* Clear second channel interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_Chnl1Flag);
/* Disable second channel interrupt.*/
FTM_DisableInterrupts(FTM0, kFTM_Chnl1InterruptEnable);
g_secondChannelOverflowCount = g_timerOverflowInterruptCount;
ftmSecondChannelInterruptFlag = true;
}
else
{}
}
```

```
void FTM_DualEdgeInputCaptureMode(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Enable dual-edge capture mode */
FTM0->COMBINE = FTM_COMBINE_DECAPEN0_MASK | FTM_COMBINE_DECAP0_MASK
| FTM_COMBINE_DECAPEN1_MASK | FTM_COMBINE_DECAP1_MASK;
/* Select positive polarity pulse width measurement and enable continuous mode for FTM0_CH0/CH2 */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_MSA_MASK | FTM_CnSC_ELSA_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK | FTM_CnSC_CHIE_MASK;
FTM0->CONTROLS[2].CnSC = FTM_CnSC_MSA_MASK | FTM_CnSC_ELSA_MASK;
FTM0->CONTROLS[3].CnSC = FTM_CnSC_ELSB_MASK | FTM_CnSC_CHIE_MASK;
/* Reset counter */
FTM0->CNT = 0;
/* Select clock */
FTM0->SC = FTM_SC_CLKS(1);
}
```

```
void FTM_PwmPulseWithCaculate(void)
{
capture1Val = FTM_GetInputCaptureValue(DEMO_FTM_BASEADDR,
(ftm_chnl_t)(BOARD_FTM_INPUT_CAPTURE_CHANNEL_PAIR * 2));
capture2Val = FTM_GetInputCaptureValue(DEMO_FTM_BASEADDR, (ftm_chnl_t)
(BOARD_FTM_INPUT_CAPTURE_CHANNEL_PAIR * 2 + 1));
/* FTM clock source is not prescaled and is divided by 1000000 as the output is printed in
 microseconds*/
pulseWidth =
(float)(((g_secondChannelOverflowCount - g_firstChannelOverflowCount) * 65536 + capture2Val -
 capture1Val) +
1) / ((float)FTM_SOURCE_CLOCK / 1000000);
}
```

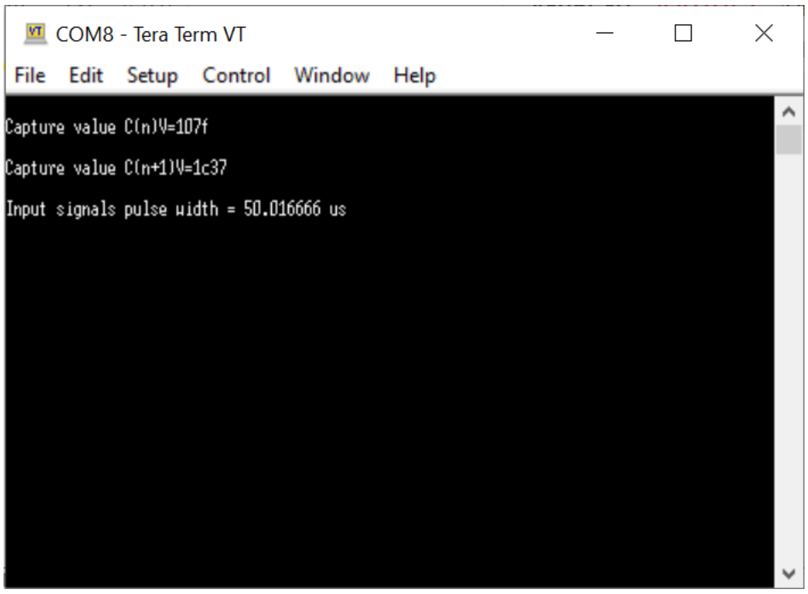
Figure 10 shows the PWM signal attached to the FTM0 used for dual edge capture. The PWM signal frequency is 10 kHz.



```
              Value      Mean     Min      Max      Std Dev
1 Frequency   10.00kHz   10.00k   9.996k   10.00k   2.102
1 Period      100.0µs    100.0µ   99.96µ   100.0µ   21.02n
1 +Width      50.00µs    50.00µ   50.00µ   50.04µ   10.01n
2 Frequency   5.556MHz   Low resolution
```

Figure 10. FTM dual-edge capture PWM input signal

Figure 11 shows the captured input PWM signal C(n)V and C(n+1)V value. This value also determines its PWM period.



```
Capture value C(n)V=107f

Capture value C(n+1)V=1c37

Input signals pulse width = 50.016666 us
```

Figure 11. FTM dual-edge capture PWM value

4.7 Quadrature decoder mode

For the motor control use case, the encoder sensor is used to detect the motor position. The Quadrature decoder mode of the FTM module can be used to decode the signals and get the position information of the motor. There are three output signals. The phase A and phase B signals consist of a series of pulses, which are phase-shifted by 90 degrees. The third signal provides the absolute position information. In the motion control, it is used to check the pulse-counting consistency. After each revolution, the value of the counted pulses is captured and compared to the defined value. If a difference is detected, the control algorithm must perform the position-offset compensation.

The LPC86x FTM module only provides the Quadrature encoder mode in the FTM1 module. The Quadrature decoder mode is enabled if QUADEN = 1. The Quadrature decoder mode uses the input signals phase A and phase B to control the FTM counter increment and decrement.

Two sub-modes can be used in the Quadrature encoder mode, which are as follows:

- Count and direction encoding mode
- Phase A and phase B encoding mode

The Count and direction encoding mode is enabled when QUADMODE = 1. In this mode, the phase A and phase B inputs imply the counting rate and the counting direction.

To process the phase A and phase B signals from the encoder sensor, enable the Phase A and phase B encoding mode (QUADMODE = 0).

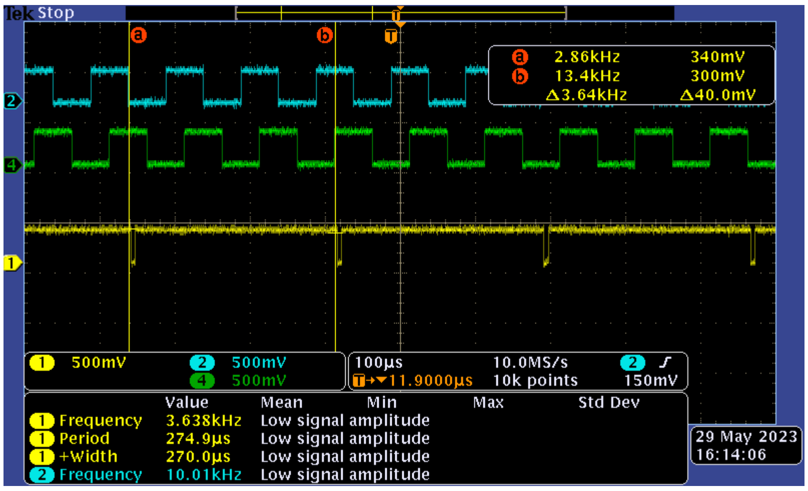
- In this mode, the phase A and phase B signals indicate the counting direction and the counting rate.
- If the phase B signal lags the phase A signal, the FTM counter increments after every detected rising/falling edge of both signals.
- If the phase B signal leads the phase A signal, the FTM counter decrements after every detected rising/falling edge of both signals. The QUADIR bit in the `FTM_QDCTRL` register indicates the counting direction.

The Quadrature decoder mode code example is as follows:

```
void FTM_QuadratureDecoderMode(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm1);
EnableIRQ(FTM1_IRQn);
/* Enable registers updating from write buffers */
FTM1->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Encoder simulation with totally 10 rising/falling edges */
FTM1->MOD = FTM_MOD_MOD(10);
FTM1->CNTIN = FTM_CNTIN_INIT(0);
FTM1->QDCTRL= FTM_QDCTRL_QUADEN_MASK;
FTM1->CNT = 0;
/* Select clock */
FTM1->SC = FTM_SC_CLKS(1) | FTM_SC_TOIE_MASK;
}
```

Figure 12 shows the following parameters:

- CH2 and CH4 in the scope represent the encoded phase A and phase B PWM signals attached to the `FTM1_QD_PHA` and `FTM1_QD_PHB`.
- CH1 in the scope represents the FTM1 counter overflow interrupt generated every time the FTM counter reaches the value of the MOD register.



	Value	Mean	Min	Max	Std Dev
1 Frequency	3.638kHz	Low signal amplitude			
1 Period	274.9µs	Low signal amplitude			
1 +Width	270.0µs	Low signal amplitude			
2 Frequency	10.01kHz	Low signal amplitude			

Figure 12. FTM quadrature decoder PWM capture

4.8 Updating the FTM registers

This section describes the flow on how to update the FTM register using the reload points, software, and hardware.

4.8.1 Update FTM registers by reload points (half cycle reload points)

The following sections describe how to update the FTM registers using the reload points.

4.8.1.1 Edge-aligned PWM register update

To change the PWM duty cycle or time period while the FTM counter is running, the half and full cycle reload strategies can be applied. This feature enables updating the FTM registers with the content of their buffers, depending on the chosen reload opportunity, by setting the LDOK bit in the `FTM_PWMLOAD` register. When a reload opportunity occurs, the RF bit in the `FTM_SC` register is set and the reload-opportunity interrupt is generated if RIE = 1. In the interrupt routine, the FTM registers can be changed and updated simultaneously.

If the Up-counting mode is selected to generate the edge-align PWM, the half and full cycle reload opportunities can update the FTM registers according to the following steps:

1. To generate the edge-align PWM and enable the FTMEN bit in the `FTM_MODE` register and the RIE bit in the `FTM_SC` register, initialize FTM0.
2. Enable the HCSEL bit in the `FTM_PWMLOAD` register.
3. For the half-cycle reload opportunity, adjust the value in the `FTM_HCR` register to MOD/2
4. For the full-cycle reload opportunity, HCSEL = 0.
5. In the interrupt routine, change the value of the FTM registers to update the values from their buffers and clear the RF bit in the `FTM_SC` register.

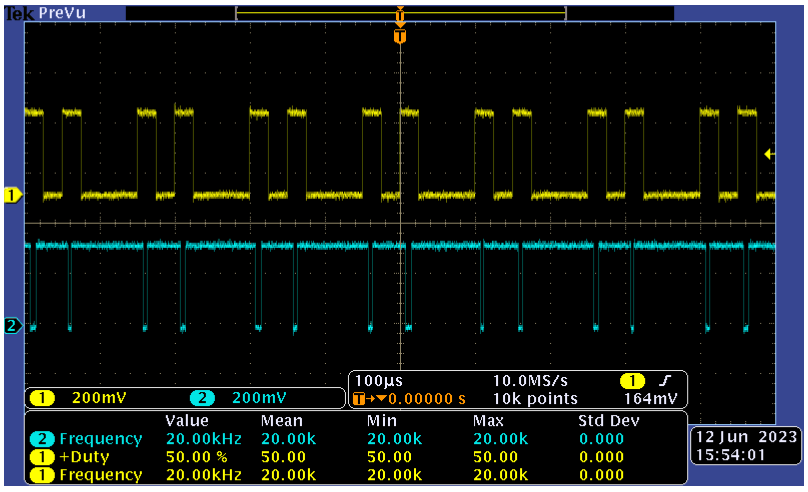
Edge-aligned PWM register update code example is as follows:

```
void FTM0_IRQHandler(void)
{
if((FTM_GetStatusFlags(FTM0) & kFTM_ReloadFlag) == kFTM_ReloadFlag)
{
if(g_flag)
{
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1);
}
else
{
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
}
g_flag = ! g_flag;
GPIO_PinInit (GPIO, 1, 20, &gpioPinConfig_1);//set PIO1_20 to Low
FTM0->PWMLOAD = FTM_PWMLOAD_LDOK_MASK;
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_ReloadFlag);
}
}
```

```
void FTM_EdgeAlignedUpdatePwmMode(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo in initialization stage (10kHz PWM frequency @60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN in initialization stage */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* Enable high-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
/* Enable high-true pulses of PWM signals */
FTM0->CONTROLS[1].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
/* Set channel value in initialization stage */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1500); // 50% duty cycle
/* Set channel value in initialization stage */
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(1500); // 50% duty cycle
/*enable HalfCycle reload*/
FTM0->PWMLOAD = FTM_PWMLOAD_HCSEL_MASK;
/*Set the halfcycle value*/
FTM0->HCR = FTM_HCR_HCVAL(3000);
/* Reset FTM counter */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_RIE_MASK;
}
```

Figure 13 shows the Edge-aligned PWM mode update MOD register with the following parameters:

- The CH1 in the scope represents the `FTM0_CH0`.
- The `FTM0_CH0` updates the MOD value each half PWM cycle.
- The CH2 in scope represents the FTM0 reload interrupt when the FTM counter C(n)V reaches MOD/2.



	Value	Mean	Min	Max	Std Dev
2 Frequency	20.00kHz	20.00k	20.00k	20.00k	0.000
1 +Duty	50.00 %	50.00	50.00	50.00	0.000
1 Frequency	20.00kHz	20.00k	20.00k	20.00k	0.000

Figure 13. FTM edge-aligned PWM MOD register update

4.8.1.2 Center-aligned PWM register update

A different scenario must be considered when the Up/Down-counting mode is selected to generate the center-aligned PWM as described in the following steps:

1. To generate the center-aligned PWM, initialize FTM0.
2. Enable the FTMEN bit in the `FTM_MODE` register and the RIE bit in the `FTM_SC` register.
3. Enable the CNTMIN and CNTMAX bits in the `FTM_SYNC` register for the half-cycle reload opportunity.
4. Enable the CNTMIN = CNTMAX = 0 for the full-cycle reload opportunity.
5. In the interrupt routine, change the value of the FTM registers to update the values from their buffers and clear the RF bit in the `FTM_SC` register.

Center-aligned PWM register update code example is as follows:

```
void FTM0_IRQHandler(void)
{
if((FTM_GetStatusFlags(FTM0) & kFTM_ReloadFlag) == kFTM_ReloadFlag)
{
FTM0->CONTROLS[0].CnV = FTM_CnV_VAL(4500);
FTM0->CONTROLS[1].CnV = FTM_CnV_VAL(4500);
GPIO_PinInit (GPIO, 1, 20, &gpioPinConfig_1);//set P101_20 to Low
FTM0->PWMLOAD = FTM_PWMLOAD_LDOK_MASK;
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_ReloadFlag);
}
}
```

```
void FTM_CenterAlignedUpdatePwmMode(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[2].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[3].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(1500); // 25% duty cycle
FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
FTM0->CONTROLS[3].CnV=FTM_CnV_VAL(1500); // 25% duty cycle
FTM0->SYNC = FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK;
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK
| FTM_SC_PWMEN2_MASK | FTM_SC_PWMEN3_MASK | FTM_SC_RIE_MASK;
}
```

Figure 14 shows the Center-aligned PWM mode update for the C(n)V and C(n+1)V register with the following parameters:

- The CH2 in the scope represents the `FTM0_CH0`.
- The `FTM0_CH0` updates the C(n)V value after the half PWM cycle.
- The CH4 in the scope represents the `FTM0_CH3`, which does not update the C(n)V value used to compare with the `FTM0_CH0`.
- The CH1 in scope represents the FTM0 reload interrupt when the FTM counter C(n)V reaches MOD/2.

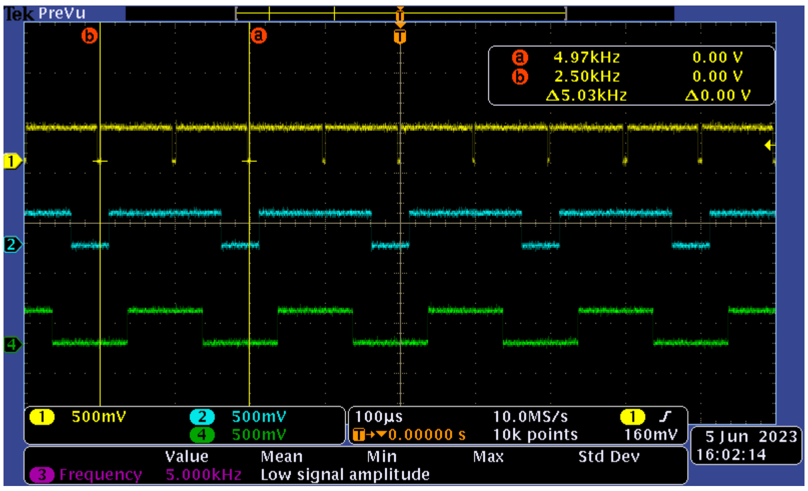


Figure 14. FTM center-aligned PWM CnV register update

4.8.2 Update FTM registers by software

The following sections describe how to update the FTM registers using the software.

4.8.2.1 Update MOD register value

The MOD register synchronization updates the MOD register with its buffer value. This synchronization is enabled if FTMEN = 1.

The MOD register synchronization is done either by the enhanced PWM synchronization (SYNCMODE = 1) or the legacy PWM synchronization (SYNCMODE = 0). However, it is expected that the MOD register is synchronized only by the enhanced PWM synchronization.

In the case of enhanced PWM synchronization, the MOD register synchronization depends on SWWRBUF, SWRSTCNT, HWWRBUF, and HWRSTCNT bits.

Figure 15 shows the MOD register update flowchart.

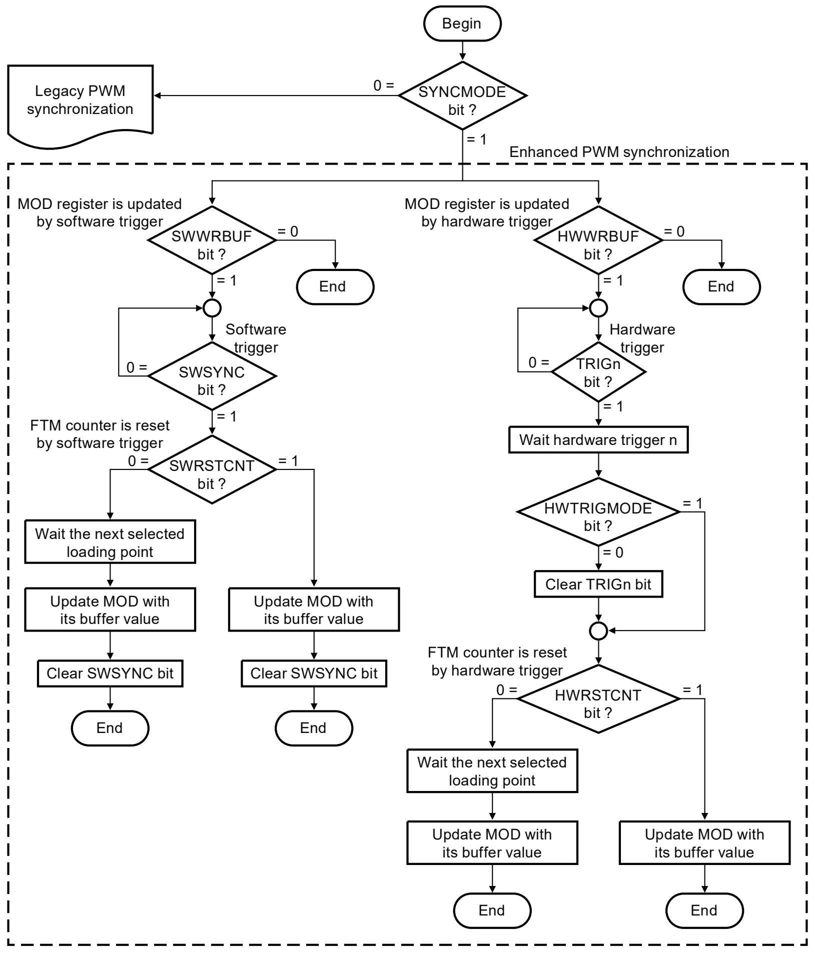

Begin
SYNCMODE bit ?
0 =
Legacy PWM synchronization
= 1
Enhanced PWM synchronization
MOD register is updated by software trigger
SWWRBUF bit ?
= 0
End
= 1
Software trigger
0 =
SWSYNC bit ?
= 1
FTM counter is reset by software trigger
0 =
SWRSTCNT bit ?
= 1
Wait the next selected loading point
Update MOD with its buffer value
Clear SWSYNC bit
End
Update MOD with its buffer value
Clear SWSYNC bit
End
MOD register is updated by hardware trigger
HWWRBUF bit ?
= 0
End
= 1
Hardware trigger
0 =
TRIGn bit ?
= 1
Wait hardware trigger n
HWTRIGMODE bit ?
= 1
= 0
Clear TRIGn bit
FTM counter is reset by hardware trigger
0 =
HWRSTCNT bit ?
= 1
Wait the next selected loading point
Update MOD with its buffer value
End
Update MOD with its buffer value
End

Figure 15. FTM MOD register update flowchart

The software MOD register updated code example is as follows:

```
void FTM0_IRQHandler(void)
{
if((FTM_GetStatusFlags(FTM0) & kFTM_ReloadFlag) == kFTM_ReloadFlag)
{
if(FTM0->MOD != FTM_MOD_MOD(3000-1))
{
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1);
GPIO_PinInit (GPIO, 1, 20, &gpioPinConfig);//set P1O1_20 to Low
/* enable software sync */
FTM0->SYNC = FTM_SYNC_SWSYNC_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
}
else
{
GPIO_PinInit (GPIO, 1, 20, &gpioPinConfig);//set P1O1_20 to Low
FTM0->SYNC = FTM_SYNC_SWSYNC_MASK;
}
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_ReloadFlag);
}
}
```

```
void FTM_CenterAlignedSwTiggerMODSYNC(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK| FTM_MODE_WPDIS_MASK;
/* Set Modulo (5kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1500); // 50% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(1500); // 50% duty cycle
FTM0->CONF = FTM_CONF_LDFQ(3);
FTM0->SYNCONF = FTM_SYNCONF_SYNCMODE_MASK | FTM_SYNCONF_SWWRBUF_MASK | FTM_SYNCONF_SWRSTCNT_MASK;
FTM0->SYNC = FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK;
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_RIE_MASK;
}
```

Figure 16 shows the result for the software MOD register update with the following parameters:

- The CH1 and CH2 in the scope represent the `FTM0_CH0`, `FTM0_CH1`.
- The CH3 represents the reload point interrupt after every three periods.
- After four periods, the reload point interrupt is generated and the MOD register can be updated in the IRQ.
- The frequency changes between 5 kHz and 10 kHz.

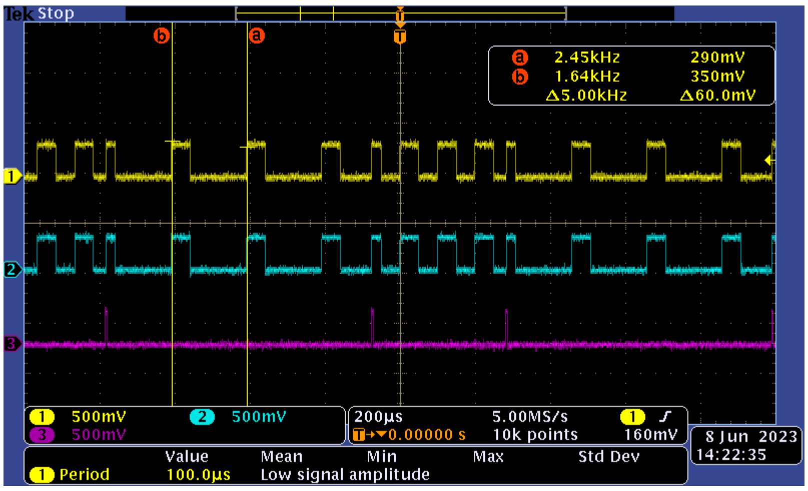


Figure 16. FTM software MOD register update

4.8.2.2 Update output mask register value

The OUTMASK register synchronization updates the OUTMASK register with its buffer value. The OUTMASK register can be updated at each rising edge of the FTM input clock (SYNCHOM = 0), by one of the following synchronizations:

- The enhanced PWM synchronization, where SYNCHOM = 1 and SYNCMODE = 1.
- The legacy PWM synchronization, where SYNCHOM = 1 and SYNCMODE = 0.
  However, it is expected that the OUTMASK register is synchronized only by the enhanced PWM synchronization.

Figure 17 shows the output mask register update flowchart.

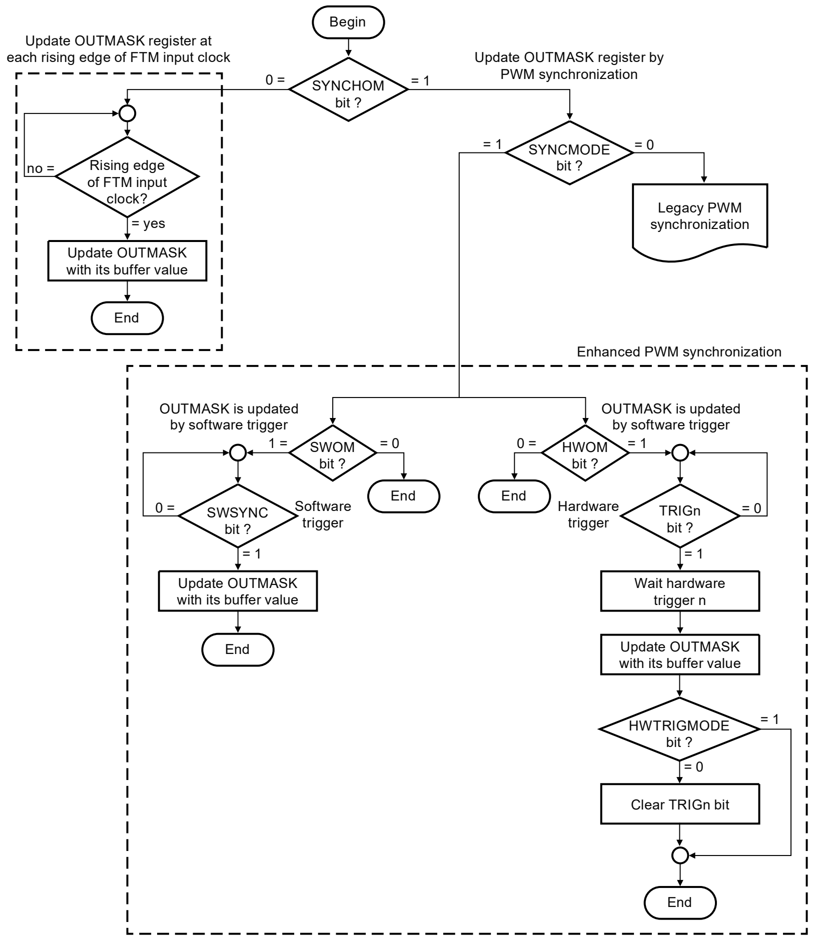

Begin
SYNCHOM bit ?
0 =
= 1
Update OUTMASK register at each rising edge of FTM input clock
Update OUTMASK register by PWM synchronization
Rising edge of FTM input clock?
no =
= yes
Update OUTMASK with its buffer value
End
SYNCMODE bit ?
= 1
= 0
Legacy PWM synchronization
Enhanced PWM synchronization
OUTMASK is updated by software trigger
SWOM bit ?
1 =
= 0
End
SWSYNC bit ?
0 =
Software trigger
= 1
Update OUTMASK with its buffer value
End
OUTMASK is updated by software trigger
HWOM bit ?
0 =
= 1
End
Hardware trigger
TRIGn bit ?
= 0
= 1
Wait hardware trigger n
Update OUTMASK with its buffer value
HWTRIGMODE bit ?
= 1
= 0
Clear TRIGn bit
End

Figure 17. FTM output mask register update flowchart

The output mask register update code example is as follows:

```
void FTM0_IRQHandler(void)
{
if((FTM_GetStatusFlags(FTM0) & kFTM_ReloadFlag) == kFTM_ReloadFlag)
{
if(!(FTM0->OUTMASK & FTM_OUTMASK_CH0OM_MASK))
{
FTM0->OUTMASK = FTM_OUTMASK_CH0OM_MASK;
GPIO_PinInit (GPIO, 1, 20, &gpioPinConfig);//set P1O1_20 to High
FTM0->SYNC = FTM_SYNC_SWSYNC_MASK;
}
else
{
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->OUTMASK &= ~(FTM_OUTMASK_CH0OM_MASK);
}
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_ReloadFlag);
}
}
```

```
void FTM_CenterAlignedSwTiggerOMSYNC(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1);
/* Set CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2250); // 75% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(2250); // 37.5% duty cycle
FTM0->SYNC = FTM_SYNC_SYNCHOM_MASK | FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK;
FTM0->CONF = FTM_CONF_LDFQ(3);
FTM0->SYNCONF = FTM_SYNCONF_SYNCMODE_MASK | FTM_SYNCONF_SWOM_MASK;
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK |FTM_SC_RIE_MASK;
}
```

Figure 18 shows the output mask register update for PWM output with the following parameters:

- CH1 and CH2 in the scope represent the `FTM0_CH1` and `FTM0_CH0`.
- CH3 represents the reload point interrupt every four periods.
- As the scope shows, the `FTM0_CH0` output mask register is updated every four periods and `FTM0_CH0` is masked after every four periods.

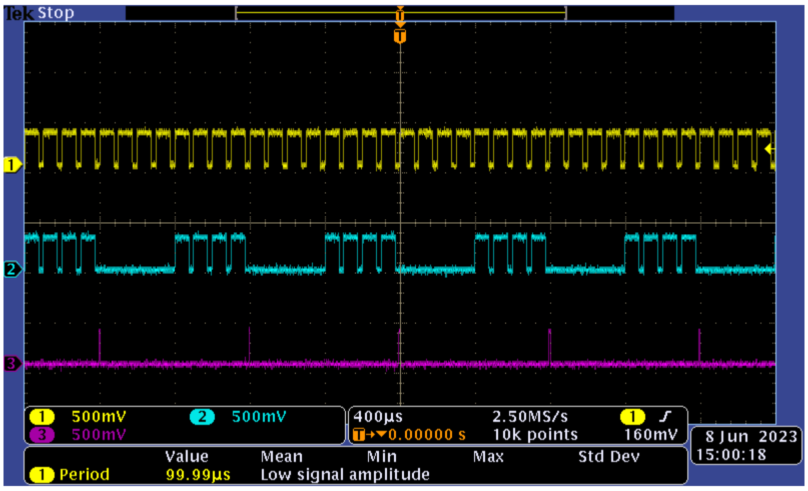


Figure 18. FTM output mask register update

4.8.2.3 Update invert register value

The INVCTRL register synchronization updates the INVCTRL register with its buffer value. The INVCTRL register can be updated at each rising edge of the FTM input clock (INVC = 0). The enhanced PWM synchronization (INVC = 1 and SYNCMODE = 1) can also update the INVCTRL register, as shown in the flowchart in Figure 19.

In the case of enhanced PWM synchronization, the INVCTRL register synchronization depends on SWINVC and HWINVC bits.

Figure 19 shows the invert register update flowchart.

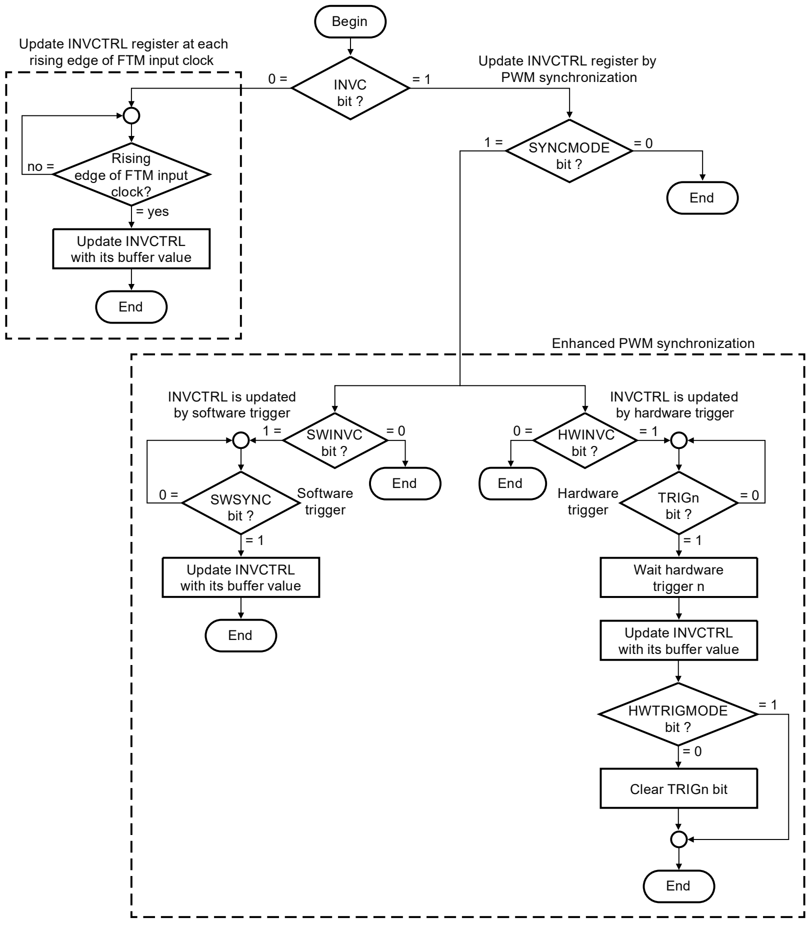

Begin
Update INVCTRL register at each rising edge of FTM input clock
0 =
INVC bit ?
= 1
Update INVCTRL register by PWM synchronization
no =
Rising edge of FTM input clock?
= yes
Update INVCTRL with its buffer value
End
1 =
SYNCMODE bit ?
= 0
End
Enhanced PWM synchronization
INVCTRL is updated by software trigger
1 =
SWINVC bit ?
= 0
End
0 =
SWSYNC bit ?
Software trigger
= 1
Update INVCTRL with its buffer value
End
INVCTRL is updated by hardware trigger
0 =
HWINVC bit ?
= 1
End
Hardware trigger
TRIGn bit ?
= 0
= 1
Wait hardware trigger n
Update INVCTRL with its buffer value
HWTRIGMODE bit ?
= 1
= 0
Clear TRIGn bit
End

Figure 19. FTM invert register update flowchart

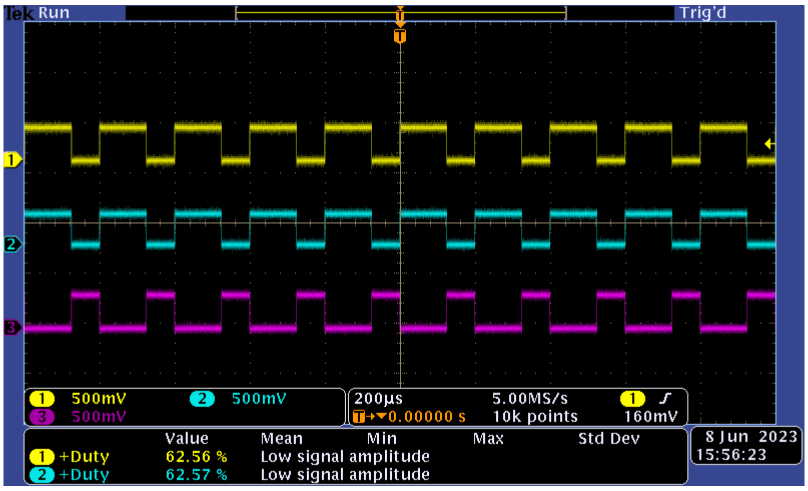
The invert register update code example is as follows:

```
void FTM0_IRQHandler(void)
{
if((FTM_GetStatusFlags(FTM0) & kFTM_ReloadFlag) == kFTM_ReloadFlag)
{
FTM0->INVCTRL = FTM_INVCTRL_INV0EN_MASK;
FTM0->SYNC = FTM_SYNC_SWSYNC_MASK;
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_ReloadFlag);
}
}
```

```
void FTM_CenterAlignedSwTiggerIVSYNC(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
EnableIRQ(FTM0_IRQn);
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[2].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2250); // 37.5% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(2250); // 37.5% duty cycle
FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(2250); // 37.5% duty cycle
FTM0->SYNC = FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK;
FTM0->INVCTRL &= ~FTM_INVCTRL_INV0EN_MASK;
FTM0->CONF = FTM_CONF_LDFQ(3);
FTM0->SYNCONF = FTM_SYNCONF_SYNCMODE_MASK | FTM_SYNCONF_SWINVC_MASK | FTM_SYNCONF_INVC_MASK;
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN2_MASK |
 FTM_SC_RIE_MASK;
}
```

Figure 20 shows the invert register update PWM output with the following parameters:

- CH1 and CH2 in the scope represent the `FTM0_CH0` and `FTM0_CH1`.
- CH3 represents `FTM0_CH2`.
- As the scope shows, `FTM0_CH3` is the PWM, which has not been inverted.
- The `FTM0_CH0` and `FTM0_CH1` represent the inverted PWM compared with `FTM0_CH3`.



```
          Value     Mean                 Min    Max    Std Dev
1 +Duty   62.56 %   Low signal amplitude
2 +Duty   62.57 %   Low signal amplitude
```

Figure 20. FTM invert register update PWM output

4.8.3 Update FTM registers by hardware

The hardware synchronization is another way of updating the FTM registers when the counter is running. For this case, the interrupt routine is not necessary because the FTM registers can be changed at any time during the code execution. Their values are updated when a hardware trigger occurs. This way the CPU load can be reduced significantly.

Three hardware trigger signal inputs of the FTM module can be selected depending on the enabled TRIGn bit in the `FTM_SYNC` register. The FlexTimer trigger input multiplexing assigns the input trigger "n" to certain signals, as shown in Figure 21.

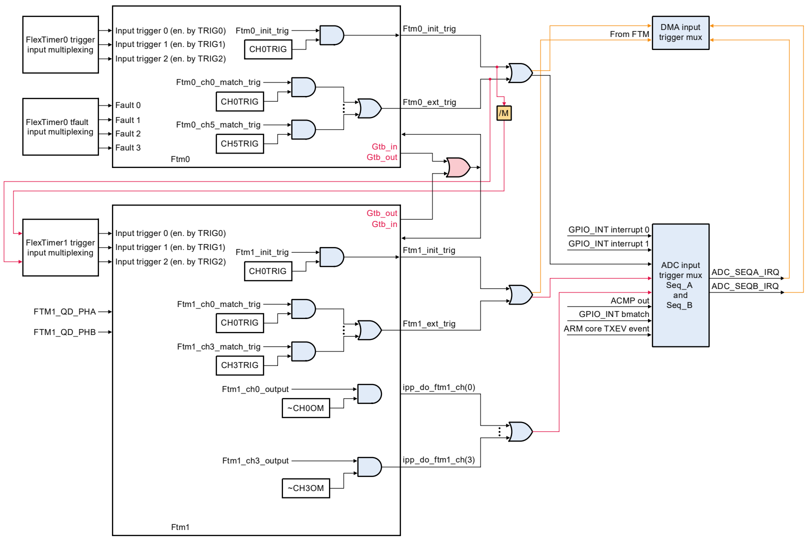


Figure 21. FTM trigger input source

Figure 22 shows the selection ID of the FlexTimer0 trigger input source. For more information on the FTM trigger input configuration, refer to the below code example.

Selection	Input source
0	FTM1_INIT_TRIG
1	FTM1_EXT_TRIG
2	ADC0_THCMP_IRQ
3	ACMP0_OUT
4	GPIOINT_BMATCH
5	ARM_TXEV
6	MRT_IRQ (global MRT interrupt)

Figure 22. FTM trigger input source selection

The hardware PWM register updated code example is as follows:

```
void FTM0_IRQHandler(void)
{
if((FTM_GetStatusFlags(FTM0) & kFTM_ReloadFlag) == kFTM_ReloadFlag)
{
if(!(FTM0->OUTMASK & FTM_OUTMASK_CH0OM_MASK))
{
FTM0->OUTMASK = FTM_OUTMASK_CH0OM_MASK;
GPIO_PinInit (GPIO, 1, 20, &gpioPinConfig);//set P1O1_20 to High
FTM0->SYNC = FTM_SYNC_SWSYNC_MASK;
}
else
{
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->OUTMASK &= ~(FTM_OUTMASK_CH0OM_MASK);
}
/* Clear overflow interrupt flag.*/
FTM_ClearStatusFlags(FTM0, kFTM_ReloadFlag);
}
}
```

```
void FTM_CenterAlignedHwTiggerUpdatePwmMode(void)
{
EnableIRQ(FTM0_IRQn);
/* FTM0 configuration*/
FTM0->SC = FTM_SC_CPWMS_MASK;
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Set Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1);
/* Set CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2250); // 62.5% duty cycle
FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(2250); // 62.5% duty cycle
/*Connect FTM1_INIT_TRIG to FTM0 trigger0*/
INPUTMUX->FTM0_INMUX[0] = 0;
/*Configure the output mask synchronization and trigger input*/
FTM0->SYNC = FTM_SYNC_SYNCHOM_MASK | FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK
| FTM_SYNC_TRIG0_MASK;
/*enable hardware trigger pwm synchronization*/
FTM0->SYNCONF = FTM_SYNCONF_SYNCMODE_MASK | FTM_SYNCONF_HWOM_MASK | FTM_SYNCONF_HWTRIGMODE_MASK;
/* FTM counter reset */
FTM0->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_RIE_MASK;
/* FTM1 configuration*/
/* Enable registers updating from write buffers */
FTM1->MODE = FTM_MODE_FTMEN_MASK;
/* Set Modulo in initialization stage (10kHz PWM frequency @60MHz system clock) */
FTM1->MOD = FTM_MOD_MOD(6000-1);
/* Set CNTIN in initialization stage */
FTM1->CNTIN = FTM_CNTIN_INIT(0);
/* Enable high-true pulses of PWM signals */
FTM1->CONTROLS[0].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
/* Set channel value in initialization stage */
FTM1->CONTROLS[0].CnV=FTM_CnV_VAL(3000); // 50% duty cycle
/*Enable FTM1 INIT_TRIG*/
FTM1->EXTTRIG = FTM_EXTTRIG_INITTRIGEN_MASK;
/* Reset FTM counter */
FTM1->CNT = 0;
/* Clock selection and enabling PWM generation */
FTM1->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
}
```

Figure 23 shows the hardware trigger invert register update for PWM output with the following parameters:

- CH1 and CH2 in the scope represent the `FTM1_CH0` and `FTM0_CH1`.
- CH3 represents `FTM0_CH0`.
- As the scope shows, `FTM1_CH0` is the hardware trigger signal.
- `FTM0_CH1` is the PWM, which has not been masked.
- The `FTM0_CH0` represents the masked PWM by hardware trigger compared with `FTM0_CH1`.

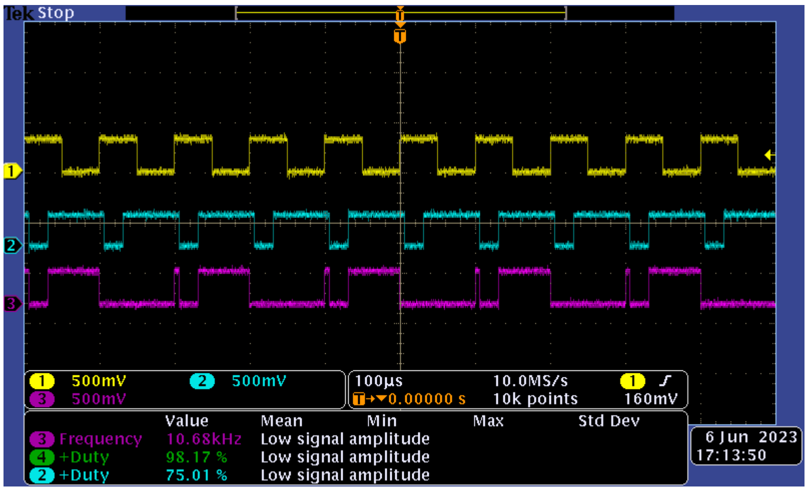

Tek Stop
1 500mV
2 500mV
3 500mV
100µs
0.00000 s
10.0MS/s
10k points
1 160mV
Value Mean Min Max Std Dev
3 Frequency 10.68kHz Low signal amplitude
4 +Duty 98.17 % Low signal amplitude
2 +Duty 75.01 % Low signal amplitude
6 Jun 2023
17:13:50

Figure 23. FTM hardware trigger output mask

4.9 Fault control and GTB feature

Fault control is an FTM feature used in motor control application.

4.9.1 Fault control

The fault control of the FTM module plays an important role in motor-control applications. It protects the power devices and the whole electrical drive system in critical moments, when undesirable behaviors such as over-temperature, over-voltage, or over-current occur. In such cases, the fault signal can be generated via a sensor or a special circuit. The fault control is able to stop all PWM channels when a fault signal is detected on the input of the FTM fault pins. An interrupt can be generated after receiving the fault signal and the undesirable behavior can be mitigated.

All FTM interrupt sources (fault interrupt, FTM counter overflow and reload opportunity interrupt, channel-compare event interrupt, and capture interrupt) share the interrupt vector. When a fault signal appears, the outputs of the FTM channels are disabled and kept at a safe logic defined in the `FTM_POL` register.

For example, if POL0 = 1 and a fault is present, `FTM_CH0` is disabled and forced to a high logic. On the contrary, if POL0 = 0 and a fault is present, `FTM_CH0` is disabled, but forced to a low logic.

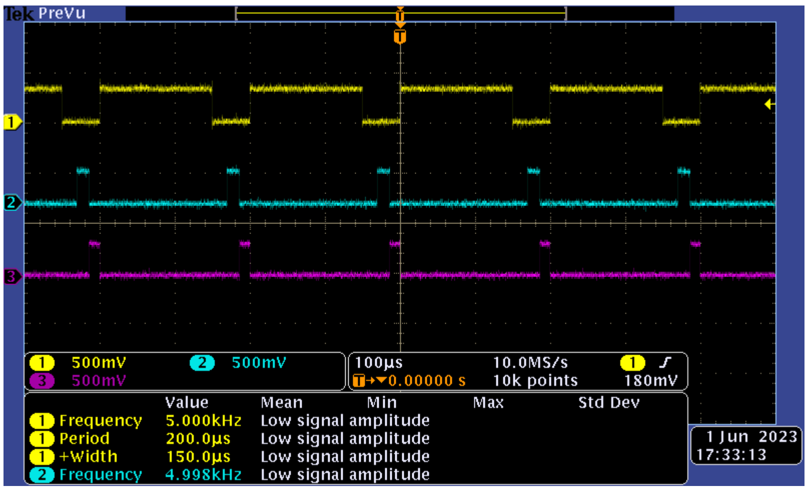
FTM has multiple channels. However, FTM cannot disable the specific channels by the means of specific fault signals. One fault signal is generated as a result of the OR operation of all entering fault signals. Whether the fault signal can disable the FTM channel or not depends on the FAULTENx bit in the `FTM_COMBINE` register. The resulting fault signal can disable all FTM channels or only the even channels (`FTM_CH0/CH2/CH4/CH6`). The selection depends on the FAULTM bit field in the `FTM_MODE` register.

Fault control code example is as follows:

```
void FTM_FaultControlCombineMode_Output(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
/* Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* Enable combine, complementary mode and dead-time for channel pair CH0/CH1*/
FTM0->COMBINE = FTM_COMBINE_COMBINE0_MASK | FTM_COMBINE_COMP0_MASK
| FTM_COMBINE_DTEN0_MASK | FTM_COMBINE_FAULTEN0_MASK;
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK; // Select high-true pulses
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK; // Select high-true pulses
/* Set Modulo (20kHz PWM frequency @60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1); // Set modulo
FTM0->CONTROLS[0].CnV = FTM_CnV_VAL(1000); // Set channel Value
FTM0->CONTROLS[1].CnV = FTM_CnV_VAL(2500); // Set channel Value
FTM0->CNT = 0; // Counter reset
FTM0->FLTCTRL = FTM_FLTCTRL_FAULT0EN_MASK;
/* Enable fault control for all channels and select automatic fault clearing mode */
FTM0->MODE |= FTM_MODE_FAULTM(0x3);
/* Safe value is set as a low after fault input is detected */
FTM0->POL = 0x0;
/* A 1 at the fault input indicates the fault */
FTM0->FLTPOL &= ~FTM_FLTPOL_FLT0POL_MASK;
FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK; // Select clock and enable PWM
}
```

Figure 24 shows the fault control output with the following parameters:

- CH1 represents the fault input signal.
- CH2 and CH3 are `FTM0_CH0` and `FTM0_CH1`.
- As the scope shows, when the fault signal is high logic, the `FTM0_CH0` and `FTM0_CH1` are masked.



```
               Value      Mean                  Min        Max        Std Dev
1 Frequency    5.000kHz   Low signal amplitude
1 Period       200.0µs    Low signal amplitude
1 +Width       150.0µs    Low signal amplitude
2 Frequency    4.998kHz   Low signal amplitude
```

Figure 24. FTM fault control output

4.9.2 Global time base

The chip supports multiple FTMs and the multiple FTM modules are independent. If the application requires more PWM channels, multiple FTM modules can be used, but they must be synchronized. The synchronization of two (or more) FTM modules means that their counters have the same values at any instant time.

LPC86x provides the global time base (GTB) mechanism to synchronize multiple FTMs. The GTB is a synchronous signal generated by the leader FTM that launches the counter of all FTMs used.

The following two conditions must be met when using the GTB function:

- Each FTM must have the same clock source
- Each FTM must start at the same time

To enable the GTB feature for each participating FTM module, perform the following steps:

1. Stop the FTM counter (write 00b to SC[CLKS]).
2. Program the FTM to the intended configuration. The FTM counter mode must be consistent across all participating modules.
3. Write 1 to CONF[GTBEEN] and write 0 to CONF[GTBEOUT] at the same time.
4. Select the intended FTM counter clock source in SC[CLKS]. The clock source must be consistent across all participating modules.
5. Reset the FTM counter (write any value to the CNT register).

The GTB code example is as follows:

```
void FTM_CenterAlignedPwmOutputGlobalBaseMode(void)
{
(void)CLOCK_EnableClock(kCLOCK_Ftm0);
(void)CLOCK_EnableClock(kCLOCK_Ftm1);
/* FTM0 Enable registers updating from write buffers */
FTM0->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* FTM1 Enable registers updating from write buffers */
FTM1->MODE = FTM_MODE_FTMEN_MASK | FTM_MODE_WPDIS_MASK;
/* FTM0 configuration*/
FTM0->SC = FTM_SC_CPWMS_MASK;
/* FTM1 configuration*/
FTM1->SC = FTM_SC_CPWMS_MASK;
/* Set FTM0 Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM0->MOD = FTM_MOD_MOD(3000-1);
/* Set FTM1 Modulo (10kHz PWM frequency at 60MHz system clock) */
FTM1->MOD = FTM_MOD_MOD(3000-1);
/* Set FTM0 CNTIN */
FTM0->CNTIN = FTM_CNTIN_INIT(0);
/* Set FTM1 CNTIN */
FTM1->CNTIN = FTM_CNTIN_INIT(0);
/* High-true pulses of PWM signals */
FTM0->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
FTM1->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
/* Set Channel Value */
FTM0->CONTROLS[0].CnV = FTM_CnV_VAL(1500); // 50% duty cycle
FTM0->CONTROLS[1].CnV = FTM_CnV_VAL(1500); // 50% duty cycle
FTM1->CONTROLS[0].CnV = FTM_CnV_VAL(1500); // 50% duty cycle
/* FTM counter reset */
FTM0->CNT = 0;
/* FTM counter reset */
FTM1->CNT = 0;
/* Enable global time base to control FTM0 and FTM1 */
FTM0->CONF = FTM_CONF_GTBEEN_MASK;
FTM1->CONF = FTM_CONF_GTBEEN_MASK;
/* FTM0 Clock selection*/
FTM0->SC |= FTM_SC_CLKS(1);
/* FTM1 Clock selection*/
FTM1->SC |= FTM_SC_CLKS(1);
/* Synchronization signal for FTM0 and FTM1 */
FTM0->CONF |= FTM_CONF_GTBEOUT_MASK;
/* FTM0 Clock selection and enabling PWM generation */
FTM0->SC |= FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
/* FTM1 Clock selection and enabling PWM generation */
FTM1->SC |= FTM_SC_PWMEN0_MASK;
}
```

Figure 25 shows the FTM output without global time base with the following parameters:

- CH1 and CH2 in scope represent the `FTM0_CH0` and `FTM0_CH1`.
- CH3 represents `FTM1_CH0`. When two FTM modules do not enable GTB function, the signals are not synchronized.

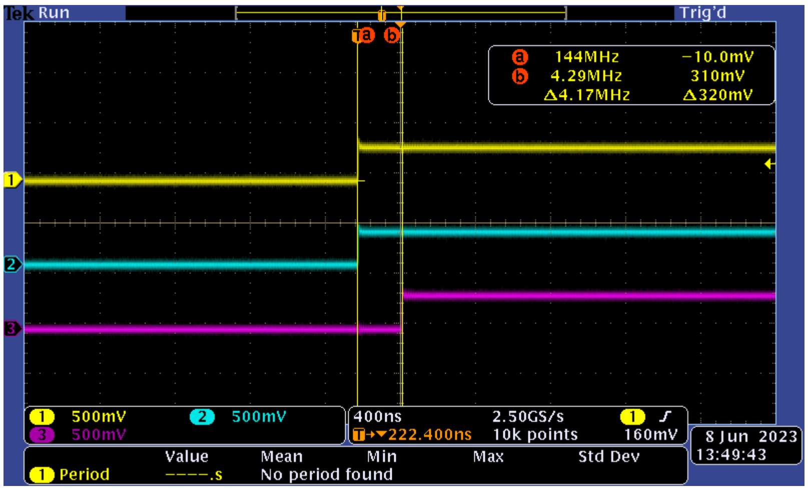


Figure 25. FTM without global time base

5 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

6 Revision history

Table 2 summarizes revisions to this document.

Table 2. Revision history

Revision history	Release date	Description
1	29 September 2023	Initial public release

7 Legal information

Contents

Date of release: 29 September 2023
Document identifier: AN14024