AN14617

Features and Operation Modes of FlexTimer Module on MCX E24x

Rev. 1.0 — 15 July 2025 Application note

Document information

Information	Content
Keywords	AN14617, MCX E24x, FlexTimer, FTM, PWM
Abstract	This application note describes the multiple features and operation modes of the FlexTimer (FTM) module on MCX E24x, and code examples and oscilloscope snapshots are provided for reference.

1 Introduction

The FlexTimer module (FTM) is an enhanced timer module that supports input capture, output compare, and the generation of PWM signals to control electric motor and power management applications and it is commonly used on many Kinetis MCUs. This application note focuses on the features of the FTM module in the MCX E24x product series, and uses the FRDM-MCXE247 to demonstrate the functionality of the FTM module.

The key features of the FTM module are as follows:

- Each channel can be configured for Input capture, Output compare, or Edge-Aligned PWM (EPWM) mode or Center-Aligned PWM (CPWM) 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 they significantly reduce the CPU load. On the other hand, the FTM module retains standard timer functions (such as the input capture or output compare modes) if it is not used for motor control. This makes FTM more flexible, covering a wide range of applications.

2 FTM overview

The FTM module has a single 16-bit counter that is used by the FTM channels for either the input or output modes. Figure 1 shows the block diagram of FTM. There are three possible clock sources that can be selected for the FTM counter: the system clock, the fixed-frequency clock, and the external clock (for a detailed clocking description, see the **Clock Distribution** chapter in the *MCX E24x reference manual*). The selected counter-clock signal then passes through a pre-scaler which enables dividing the clock source by up to 128 by setting `PS[2:0]` in the `FTM_SC` register. If the CPWMS bit in the `FTM_SC` register is disabled, the FTM counter operates in the up-counting mode. Otherwise, it operates in the up/down-counting mode.

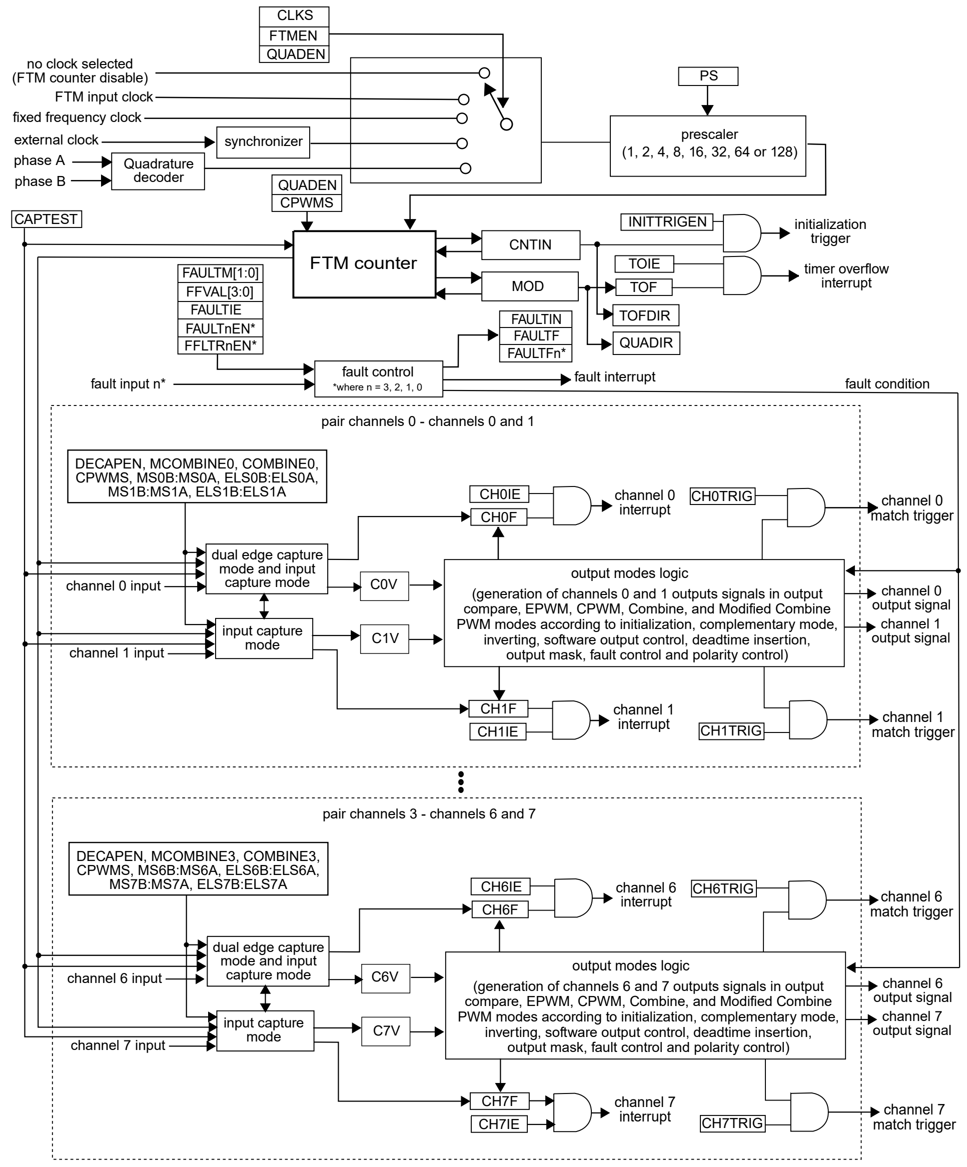

CLKS
FTMEN
QUADEN
no clock selected (FTM counter disable)
FTM input clock
fixed frequency clock
external clock
synchronizer
phase A
phase B
Quadrature decoder
PS
prescaler (1, 2, 4, 8, 16, 32, 64 or 128)
QUADEN
CPWMS
CAPTEST
FTM counter
CNTIN
MOD
INITTRIGEN
initialization trigger
TOIE
TOF
timer overflow interrupt
TOFDIR
QUADIR
FAULTM[1:0]
FFVAL[3:0]
FAULTIE
FAULTnEN*
FFLTRnEN*
FAULTIN
FAULTF
FAULTFn*
fault control *where n = 3, 2, 1, 0
fault input n*
fault interrupt
fault condition
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
input capture mode
channel 1 input
C1V
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, fault control and polarity control)
channel 0 output signal
channel 1 output signal
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
input capture mode
channel 7 input
C7V
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, fault control and polarity control)
channel 6 output signal
channel 7 output signal
CH7F
CH7IE
channel 7 interrupt
CH7TRIG
channel 7 match trigger

Figure 1. FTM block diagram

The current implementation of the MCX E24x device is equipped with up to eight FTMs. Each FTM module has eight channels which work either independently or in pairs (CH0/CH1, CH2/CH3, CH4/CH5, and CH6/CH7). All channels can be configured for the input capture, output compare, or EPWM/CPWM modes. In the EPWM mode, the FTM channel pairs can work in the complementary or/and combine modes. In addition, FTM1 and FTM2 are able to work in the quadrature-decoder mode that enables processing signals from the incremental rotary position sensor (encoder).

Table 1 shows the mode selection of channels.

Table 1. FTM channel mode setting

DECAPEN	MCOMBINE	COMBINE	CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
0	0	0	0	00	01	Input capture	Capture on rising edge only
0	0	0	0	00	10	Input capture	Capture on falling edge only
0	0	0	0	00	11	Input capture	Capture on rising or falling edge
0	0	0	0	01	01	Output compare	Toggle output on match
0	0	0	0	01	10	Output compare	Clear output on match
0	0	0	0	01	11	Output compare	Set output on match
0	0	0	0	1X	10	EPWM	High-true pulses (clear output on match)
0	0	0	0	1X	X1	EPWM	Low-true pulses (set output on match)
0	0	0	1	XX	10	CPWM	High-true pulses (clear output on match-up)
0	0	0	1	XX	X1	CPWM	Low-true pulses (set output on match-up)
0	0	1	0	XX	10	Combine PWM	High-true pulses (set on channel (n) match, and clear on channel (n +1) match)
0	0	1	0	XX	X1	Combine PWM	Low-true pulses (clear on channel (n) match, and set on channel (n+1) match)
0	1	0	0	XX	10	Modified combine PWM	High-true pulses (set on channel (n) match, and clear on channel (n +1) match)
0	1	0	0	XX	X1	Modified combine PWM	Low-true pulses (clear on channel (n) match, and set on channel (n+1) match)
1	0	0	0	X0	See Table 2	Dual -edge capture mode	One-shot capture mode
1	0	0	0	X1	See Table 2	Dual -edge capture mode	Continuous capture mode

Table 2. Dual edge capture mode—edge polarity selection

ELSnB	ELSnA	Channel port enable	Detected edges
0	0	Disabled	No edge
0	1	Enabled	Rising edge
1	0	Enabled	Falling edge
1	1	Enabled	Rising and falling edges

In sensor-based motor-control applications, it is necessary to use two FTM modules. The first FTM is used to generate six PWM signals to control the acceleration of the motor and the second FTM module works in the quadrature-decoder mode to determine the position from two 90°-shifted encoder signals, Phase A and Phase B. If the Hall sensors are used instead of the encoder, the second FTM module is configured for the input capture mode and the PWM signals of the first FTM module are controlled according to the Hall sensor logic. For more details, see the *BLDC motor control w/ Hall Effect sensor - Kinetis MCUs* (document AN4376).

3 FTM operation modes

This chapter describes the main operation modes of FTM, including:

- Section 3.1 "EPWM mode"
- Section 3.2 "CPWM mode"
- Section 3.3 "Complementary mode and dead-time insertion"
- Section 3.4 "Combine mode"
- Section 3.5 "Single-edge capture mode"
- Section 3.6 "Dual-edge capture mode"
- Section 3.7 "Quadrature decoder mode"

3.1 EPWM mode

In the EPWM mode, the FTM counter counts up from the `FTM_CNTIN` value to the `FTM_MOD` value. Signals of all FTM channels align at the edge when the FTM counter changes from the MOD value to the CNTIN value.

The EPWM mode is selected when:

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

The EPWM period is determined by MOD – CNTIN + 0x0001 and the pulse width (or the duty cycle) is determined by CnV – CNTIN or MOD – CNTIN – CnV, depending on the control bits `ELSnB:ELSnA`.

The core code for EPWM mode is as follows:

```
void Edge_Align_PWM_Init()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
```

```
    /* Set PORT pins for FTM0 */
    PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
    PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
    PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
    PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
    /* Enable complement mode and dead-time for pair channel 0/1 and 4/5 */
    FTM0->COMBINE = FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK
                  | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_DTEN2_MASK;
    /* Set Modulo in initialization stage (10kHz PWM frequency @48MHz system clock) */
    FTM0->MOD = FTM_MOD_MOD(4800-1);
    /* 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;
    FTM0->CONTROLS[4].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
    FTM0->CONTROLS[5].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
    /* Set channel value in initialization stage */
    FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2400); // 50% duty cycle
    FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(1200); // 25% duty cycle
    /* Reset FTM counter */
    FTM0->CNT = 0;
    /* Insert deadtime (1us) */
    FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(3);
    /* Clock selection and enabling PWM generation */
    FTM0->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK
             | FTM_SC_PWMEN5_MASK;
}
```

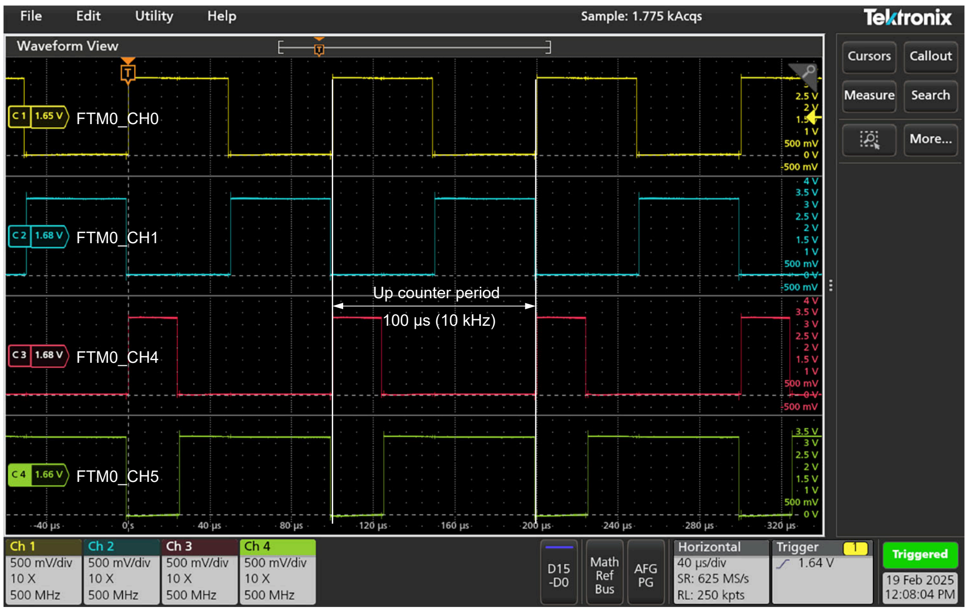


Figure 2. EPWM

Figure 2 shows the EPWM mode signal output:

- The scope channels CH1, CH2, CH3, and CH4 represent `FTM0_CH0`, `FTM0_CH1`, `FTM0_CH4`, and `FTM0_CH5` of the FTM0 module, respectively.
- The first channel pair (`FTM0_CH0`/`FTM0_CH1`) and the second channel pair (`FTM0_CH4`/`FTM0_CH5`) work in the complementary mode with 50 % and 25 % duty cycles, respectively. The second channel pair (`FTM0_CH4`/`FTM0_CH5`) demonstrates the edge alignment.

3.2 CPWM mode

In the CPWM mode, the FTM counter counts up from `FTM_CNTIN` to `FTM_MOD` and then counts down from `FTM_MOD` to `FTM_CTNIN`. Signals of all FTM channels align at the point where the FTM counter reaches the `FTM_MOD` value. The CPWM mode is selected when:

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

The duty cycle of the CPWM is determined as 2 × (CnV − CNTIN). The period is determined as 2 × (MOD − CNTIN).

The core code for CPWM mode is as follows:

```
void Center_Align_PWM_Init()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
  PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
  PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
  /* Select up-down counter for Center-Align PWM */
  FTM0->SC = FTM_SC_CPWMS_MASK;
  /* Complement mode and dead-time enable for channel0 and channel1 */
  FTM0->COMBINE = FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK;
  /* Complement mode and dead-time enable for channel4 and channel5 */
  FTM0->COMBINE |= FTM_COMBINE_COMP2_MASK | FTM_COMBINE_DTEN2_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(2400-1);
  /* 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[4].CnSC = FTM_CnSC_ELSB_MASK;
  FTM0->CONTROLS[5].CnSC = FTM_CnSC_ELSB_MASK;
  /* Set Channel Value */
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // 50% duty cycle
  FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(600); // 25% duty cycle
  /* FTM counter reset */
  FTM0->CNT = 0;
  /* Insert deadtime (1us) */
  FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) |  FTM_DEADTIME_DTVAL(3);
  /* Clock selection and enabling PWM generation */
  FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK
            | FTM_SC_PWMEN5_MASK;
}
```

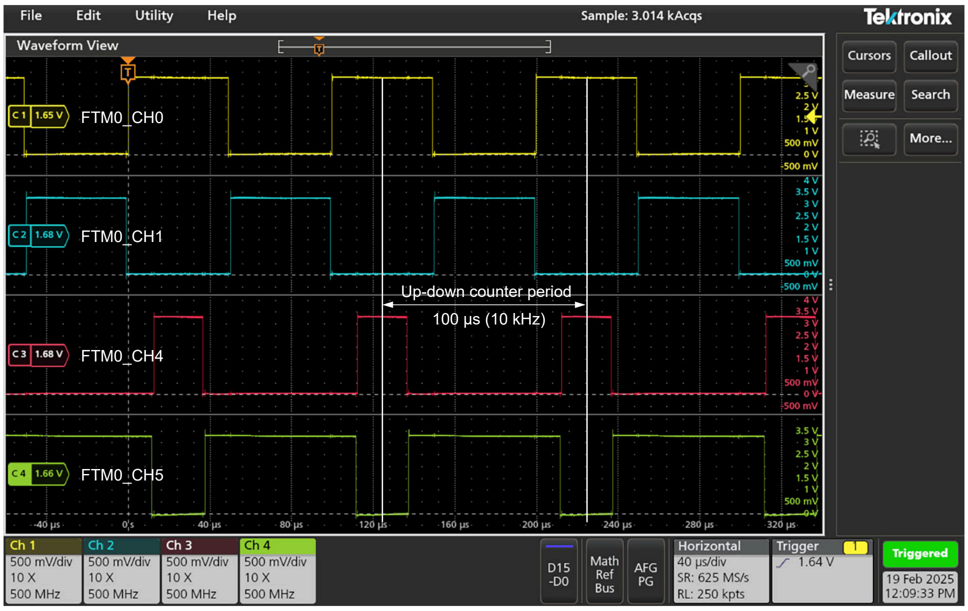


Figure 3. CPWM

Figure 3 shows the CPWM mode signal output:

- The scope channels CH1, CH2, CH3, and CH4 represent `FTM0_CH0`, `FTM0_CH1`, `FTM0_CH4`, and `FTM0_CH5` of the `FTM0` module, respectively.
- The first channel pair (`FTM0_CH0`/`FTM0_CH1`) and the second channel pair (`FTM0_CH4`/`FTM0_CH5`) work in the complementary mode with 50 % and 25 % duty cycles, respectively. The second channel pair (`FTM0_CH4`/`FTM0_CH5`) demonstrates the center alignment.

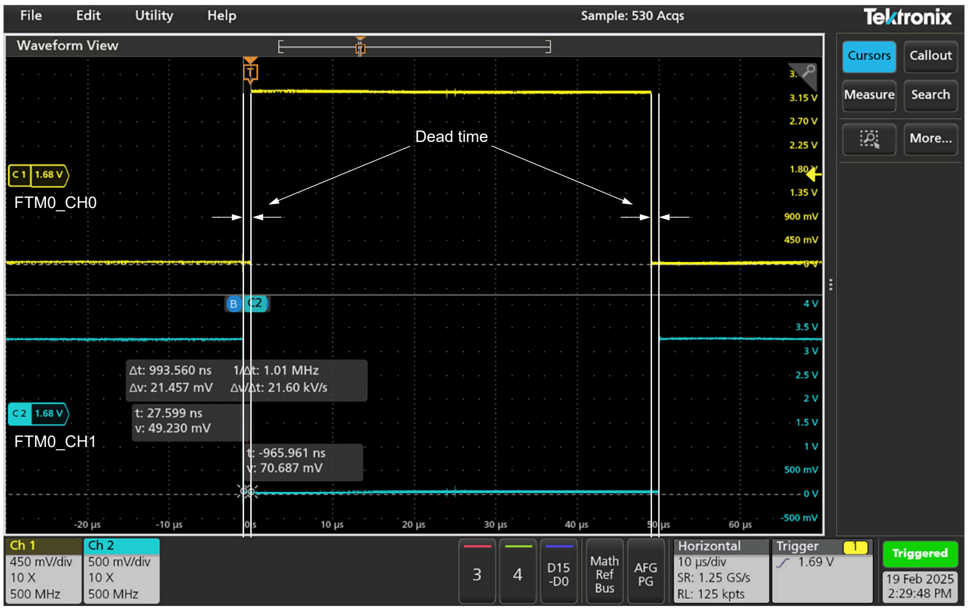
Note: *The up/down-counting mode is dedicated for generating the CPWM. The up-counting mode can be used as well. However, the FTM channels must be configured to work in the combine mode. For more information, see Section 3.4.*

3.3 Complementary mode and dead-time insertion

The FTM module supports the complementary mode. If the complementary mode is enabled by the COMP bit in the `FTM_COMBINE` register, the output signal is generated by the even FTM channel only. The odd output signal is generated by the complementary logic 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. This feature can be enabled by the `DTEN` bit in the `FTMxCOMBINEm` 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. The first two most significant bits `DTPS[1:0]` define the pre-scaler of the system clock. The bits `DTVAL[5:0]` define the duty-cycle value using the pre-scaled clock.

The complementary mode and dead-time insertion are applied to the EPWM and CPWM modes, described in the previous sections. Figure 4 shows the enlarged view of previous EPWM. A dead time of 1 µs is inserted into the PWM signal.



aaa-060030

Figure 4. Complementary mode of channel pair `FTM0_CH0`/`FTM0_CH1` with dead-time insertion

3.4 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, where 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. For more details, see the *3-phase Sensorless BLDC Motor Control Reference Design Using Kinetis KEA128* (document DRM151).

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. The core code for Phase-shifted PWM mode is as follows:

```
void Phase_Shifted_PWM()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
```

```
    PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
    PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
    PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
    /* Enable combine, complementary mode and dead-time for channel pair CH0/CH1 and CH4/CH5*/
    FTM0->COMBINE = FTM_COMBINE_COMBINE0_MASK | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK
                  | FTM_COMBINE_COMBINE2_MASK | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_DTEN2_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
    FTM0->CONTROLS[4].CnSC=FTM_CnSC_ELSB_MASK; // Select high-true pulses
    FTM0->CONTROLS[5].CnSC=FTM_CnSC_ELSB_MASK; // Select high-true pulses
    /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
    FTM0->MOD = FTM_MOD_MOD(4800-1); // Set modulo
    FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // Set channel Value
    FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(3600); // Set channel Value
    FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(2400); // Set channel Value
    FTM0->CONTROLS[5].CnV=FTM_CnV_VAL(4800); // Set channel Value
    FTM0->CNT = 0; // Counter reset
    /* Insert deadtime (1us) */
    FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(3);
    FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK
              | FTM_SC_PWMEN5_MASK; // Select clock and enable PWM
  }
```

The resulting behavior of the code example is as shown in Figure 5.

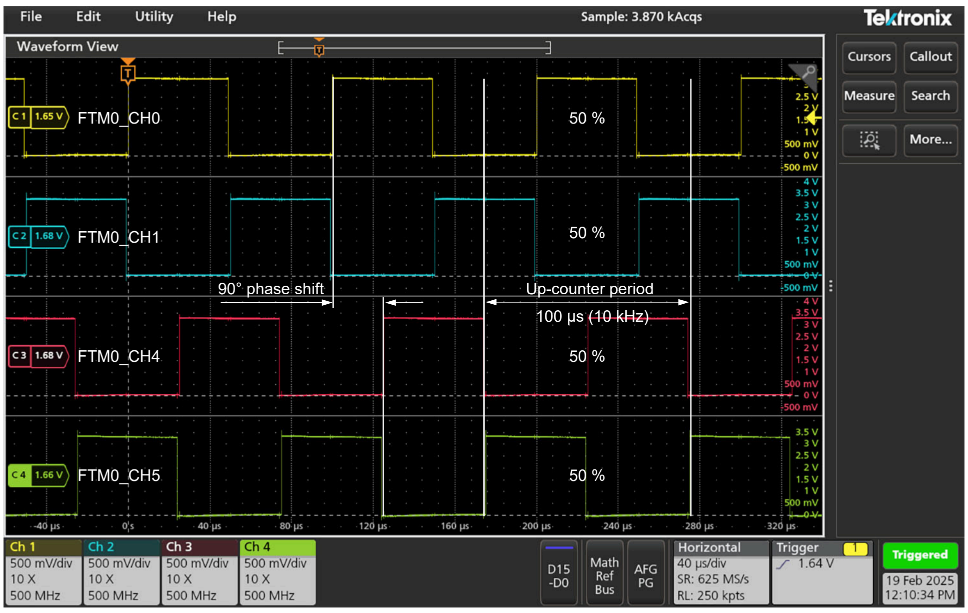


Figure 5. Phase-shifted PWM

Figure 5 shows the phase-shifted PWM mode signal output:

- The scope channels CH1, CH2, CH3, and CH4 represent the FTM0 module channels `FTM0_CH0`, `FTM0_CH1`, `FTM0_CH4`, and `FTM0_CH5`, respectively.
- The first channel pair (`FTM0_CH0`/`FTM0_CH1`) and the second channel pair (`FTM0_CH4`/`FTM0_CH5`) both work in the complementary mode with a 50 % duty cycle. The second channel pair (`FTM0_CH4`/`FTM0_CH5`) is phase-shifted by 90° to the first channel pair (`FTM0_CH0`/`FTM0_CH1`).

Figure 5 shows the flexibility of the PWM generation because the 50 % duty cycle can be edge-aligned, center-aligned, or variously shifted according to the specific application requirements.

3.5 Single-edge capture mode

The FTM capture mode is used mostly to determine the pulse width or the period of the tested signal. Another option of the FTM capture mode is to detect the rising/falling edge of an external signal and to generate an interrupt to notify that an external event appeared. The FTM capture mode is also used in BLDC motor-control applications, where the Hall sensors are used to detect the position of the rotor and to 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 generate 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, select the input channel of the FTM module `FTM_CHx` and select the edge-sensitive input by the control bits `ELSnB:ELSnA`. When the selected edge occurs on the channel input, the current value of the FTM counter is captured in the `CnV` register and a channel interrupt is generated (if CH(n)IE = 1). In the interrupt routine, save the value of the `CnV` register into a variable and make 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 on 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.

This example shows the input capture mode of `FTM0` that measures the time period of the input signal through the `FTM0_CH0` channel.

```
void FTM0_Single_Edge_Capture_Mode()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // Set PTD15 for FTM0 - Channel0

  EnableIRQ(FTM0_Ch0_Ch1_IRQn); // Enable interrupt
  /* 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);
}

void FTM1_PWM_Output()
{
  /* Enable clock for PORTA */
  CLOCK_EnableClock(kCLOCK_PortA);
  /* Enable clock for FTM1 */
  CLOCK_EnableClock(kCLOCK_Ftm1);
  /* Enable registers updating from write buffers */
  FTM1->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM1 */
  PORTA->PCR[16] = PORT_PCR_MUX(2); // FTM1, Channel3

  /* Set Modulo in initialization stage (10kHz PWM frequency @48MHz system clock) */
  FTM1->MOD = FTM_MOD_MOD(4800-1);
  /* Set CNTIN in initialization stage */
  FTM1->CNTIN = FTM_CNTIN_INIT(0);
  /* Enable high-true pulses of PWM signals */
```

```
    FTM1->CONTROLS[3].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
    /* Set channel value in initialization stage */
    FTM1->CONTROLS[3].CnV=FTM_CnV_VAL(2400); // 50% duty cycle

    /* Reset counter */
    FTM1->CNT = 0;
    /* Select clock */
    FTM1->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN3_MASK;
}

void FTM0_Ch0_Ch1_IRQHandler()
{
    FTM0_CH0_period = FTM0->CONTROLS[0].CnV - temp; // Period calculation
    temp = FTM0->CONTROLS[0].CnV; // Save C0V value into the variable
    FTM0->CONTROLS[0].CnSC &= ~FTM_CnSC_CHF_MASK; // clear channel flag
}
```

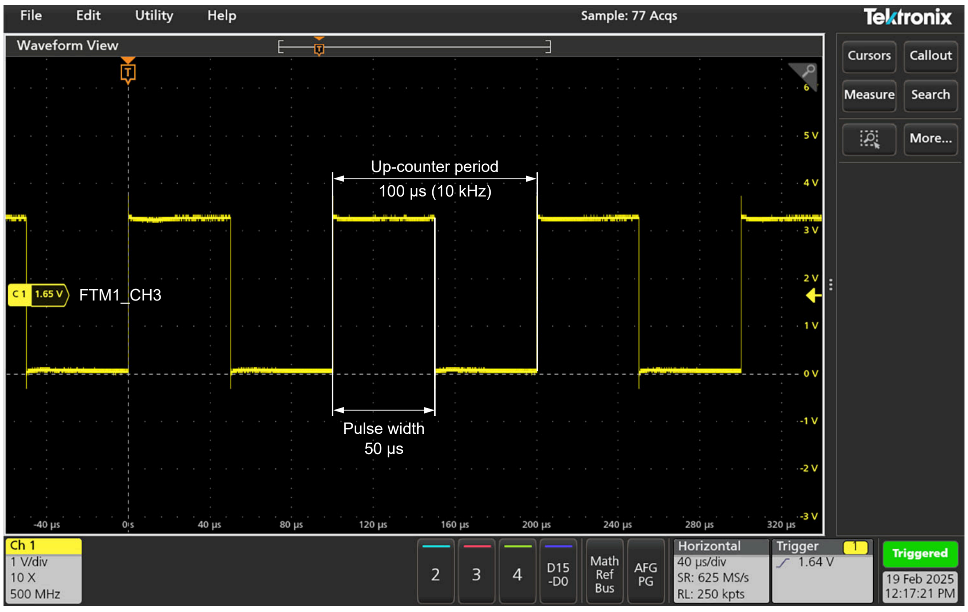
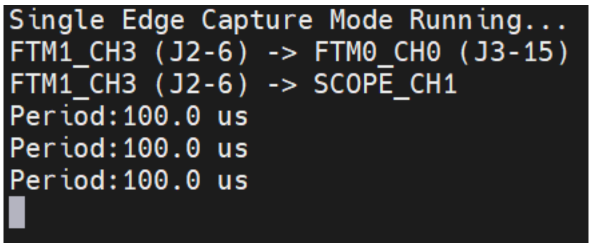


Figure 6. `FTM0_CH0` **input signal in single edge capture mode**

Figure 6 shows the single edge capture mode signal output:

- The scope channel CH1 represents the tested signal that is input to the `FTM0_CH0` channel (generated from `FTM1_CH3`).
- Every time a rising edge occurs, the capture interrupt is generated and the period of the signal is calculated and printed through UART, as shown in Figure 7. The period of the tested signal is 100 µs.



```
Single Edge Capture Mode Running...
FTM1_CH3 (J2-6) -> FTM0_CH0 (J3-15)
FTM1_CH3 (J2-6) -> SCOPE_CH1
Period:100.0 us
Period:100.0 us
Period:100.0 us
```

Figure 7. Period of tested signal printed through UART

3.6 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, the signals must be input through the even FTM channels, while the odd channels are ignored.

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. The edges are captured if the `DECAP` bit is enabled. For every new measurement, `CH(n)F` and `CH(n+1)F` must be cleared and the `DECAP` bit must be set again. The continuous capture mode is selected when MS(n)A = 1. In this mode, the edges are captured continuously if the `DECAP` bit is set. 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 modes), configure channel (n) to capture the rising edge (ELS(n)B:ELS(n)A = 1:0) and channel (n+1) to capture the falling edge (ELS(n+1)B:ELS(n+1)A = 0:1). When a second falling edge of the tested signal is detected, `CH(n+1)F` is set, the `DECAP` bit is cleared, and an interrupt is generated (if CH(n+1)IE=1). In the interrupt routine, make a subtraction of the values saved in the `C(n+1)V` and `C(n)V` registers to determine the positive-polarity pulse width of the tested signal and clear the `CH(n+1)F` bit.

If the application requires to measure the negative-polarity pulse width of the tested signal, configure channel (n) to capture the falling edge (ELS(n)B:ELS(n)A = 0:1) and channel (n+1) to capture the rising edge (ELS(n+1)B:ELS(n+1)A = 1:0). To determine the period of the tested signal, channel (n) and channel (n+1) must be sensitive on the same edges.

This example shows the dual-edge capture mode of the `FTM0` module that is used to determine the positive pulse width of the input signals.

```c
void FTM0_Dual_Edge_Capture_Mode()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Set PORTD pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  EnableIRQ(FTM0_Ch0_Ch1_IRQn); // Enable interrupt
  /* Enable dual-edge capture mode */
  FTM0->COMBINE = FTM_COMBINE_DECAPEN0_MASK | FTM_COMBINE_DECAP0_MASK;
  /* Select positive polarity pulse width measurement and enable continuous mode for FTM0_CH0 */
  FTM0->CONTROLS[0].CnSC = FTM_CnSC_MSA_MASK | FTM_CnSC_ELSA_MASK;
  FTM0->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK | FTM_CnSC_CHIE_MASK;
  /* Reset counter */
  FTM0->CNT = 0;
  /* Select clock */
  FTM0->SC = FTM_SC_CLKS(1);
}

void FTM1_PWM_Output()
{
  /* Enable clock for PORTA */
  CLOCK_EnableClock(kCLOCK_PortA);
  /* Enable clock for FTM1 */
  CLOCK_EnableClock(kCLOCK_Ftm1);
  /* Enable registers updating from write buffers */
  FTM1->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM1 */
  PORTA->PCR[16] = PORT_PCR_MUX(2); // FTM1, Channel3

  /* Set Modulo in initialization stage (10kHz PWM frequency @48MHz system clock) */
  FTM1->MOD = FTM_MOD_MOD(4800-1);
  /* Set CNTIN in initialization stage */
  FTM1->CNTIN = FTM_CNTIN_INIT(0);
  /* Enable high-true pulses of PWM signals */
  FTM1->CONTROLS[3].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
  /* Set channel value in initialization stage */
  FTM1->CONTROLS[3].CnV=FTM_CnV_VAL(2400); // 50% duty cycle
  /* Reset counter */
  FTM1->CNT = 0;
  /* Select clock */
  FTM1->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN3_MASK;
}

void FTM0_Ch0_Ch1_IRQHandler()
{
  CH0_edge0 = FTM0->CONTROLS[0].CnV; // record C0V value into the variable
  CH0_edge1 = FTM0->CONTROLS[1].CnV; // record C1V value into the variable
  FTM0_CH0_pulse_width = CH0_edge1 - CH0_edge0; // pulse width calculation of FTM0_CH0 signal
  FTM0->CONTROLS[1].CnSC &= ~FTM_CnSC_CHF_MASK; // clear FTM0_CH1 capture flag
}
```

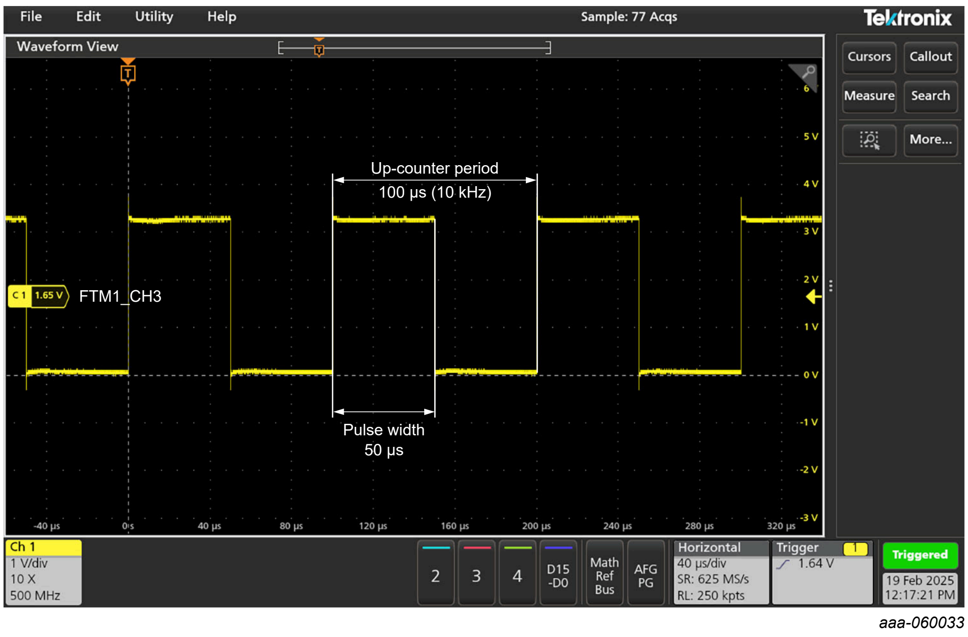
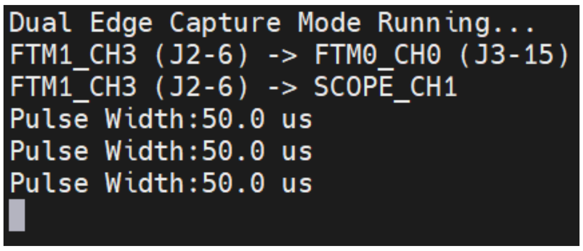


Figure 8. **`FTM0_CH0` input signal in dual edge capture mode**

Figure 8 shows the dual edge capture mode signal output:

- The scope channel CH1 represents the tested signal that is input to the `FTM0_CH0` channel (generated from `FTM1_CH3`).
- The pulse width of the signal is calculated and printed through UART, as shown in Figure 9. The pulse width of the tested signal is 50 µs.



```
Dual Edge Capture Mode Running...
FTM1_CH3 (J2-6) -> FTM0_CH0 (J3-15)
FTM1_CH3 (J2-6) -> SCOPE_CH1
Pulse Width:50.0 us
Pulse Width:50.0 us
Pulse Width:50.0 us
```

Figure 9. Positive-polarity pulse width of tested signal printed through UART

3.7 Quadrature decoder mode

The quadrature decoder mode of the FTM module is used to decode the signals from the incremental encoder sensor which is commonly used in motor-control applications for the position measurement. The incremental encoders are based on the optical technology.

The encoder has three output signals. The Phase A and Phase B signals consist of a series of pulses which are phase-shifted by 90° (therefore the term “quadrature” is used). The third signal (called “Index”) 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 current implementation of the MCX E24x device is equipped with up to eight FTM modules. While all FTMs can be configured to generate PWM signals to control electric motors, only the FTM1 and FTM2 modules can work in the quadrature decoder mode to measure the rotor position. They have dedicated pins for the Phase A and Phase B signals.

The quadrature encoder mode is selected when QUADEN = 1. There are two sub-modes that can be used in the quadrature encoder mode: the count and direction encoding mode and the Phase A and Phase B encoding mode. The count and direction encoding mode is enabled when QUADMODE = 1. In this mode, the Phase B input indicates the counting direction and the Phase A input defines the counting rate.

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 and the `QUADIR` bit in the `FTM_QDCTRL` register indicates the counting direction.

This code example demonstrates the functionality of the quadrature decoder mode with the activated Phase A and Phase B encoding sub-mode. `FTM0` is used to generate PWM signals to simulate encoder signals and `FTM1` to implement the quadrature decoder function.

```
void FTM1_Quadrature_Decoder_Mode()
{
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for PORTC*/
  CLOCK_EnableClock(kCLOCK_PortC);
  /* Enable clock for PORTE*/
  CLOCK_EnableClock(kCLOCK_PortE);
  /* Enable clock for FTM1*/
  CLOCK_EnableClock(kCLOCK_Ftm1);
  /* Set PORT pins for FTM1 */
  PORTC->PCR[7] = PORT_PCR_MUX(6); // Set PTC7 for FTM1 - Phase A input
  PORTB->PCR[2] = PORT_PCR_MUX(4); // Set PTB2 for FTM1 - Phase B input
  PORTE->PCR[7] = PORT_PCR_MUX(1); // Set PTE7 for GPIO (FTM1 overflow flag)

  /* GPIO Initialize */
  gpio_pin_config_t gpio_config =
  {
   kGPIO_DigitalOutput,
   0
  };
  GPIO_PinInit(GPIOE, 7, &gpio_config); // PTE7 init
  EnableIRQ(FTM1_Ovf_Reload_IRQn); // Enable FTM1 overflow interrupt

  /* 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;
}
```

```
void FTM0_Encoder_Output()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
  PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
  PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
  /* Enable combine, complementary mode for channel pair CH0/CH1 and CH4/CH5 */
  FTM0->COMBINE = FTM_COMBINE_COMBINE0_MASK | FTM_COMBINE_COMP0_MASK
              | FTM_COMBINE_COMBINE2_MASK | FTM_COMBINE_COMP2_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
  FTM0->CONTROLS[4].CnSC=FTM_CnSC_ELSB_MASK; // Select high-true pulses
  FTM0->CONTROLS[5].CnSC=FTM_CnSC_ELSB_MASK; // Select high-true pulses
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(4800-1); // Set modulo
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // Set channel Value
  FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(3600); // Set channel Value
  FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(2400); // Set channel Value
  FTM0->CONTROLS[5].CnV=FTM_CnV_VAL(4800); // Set channel Value
  FTM0->CNT = 0; // Counter reset
  FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK
          | FTM_SC_PWMEN5_MASK; // Select clock and enable PWM
}

void FTM1_Ovf_Reload_IRQHandler()
{
  GPIO_PortToggle(GPIOE, 1 << 7); // Show interrupt
  FTM1->SC &= ~FTM_SC_TOF_MASK;   // Clear flag
}
```

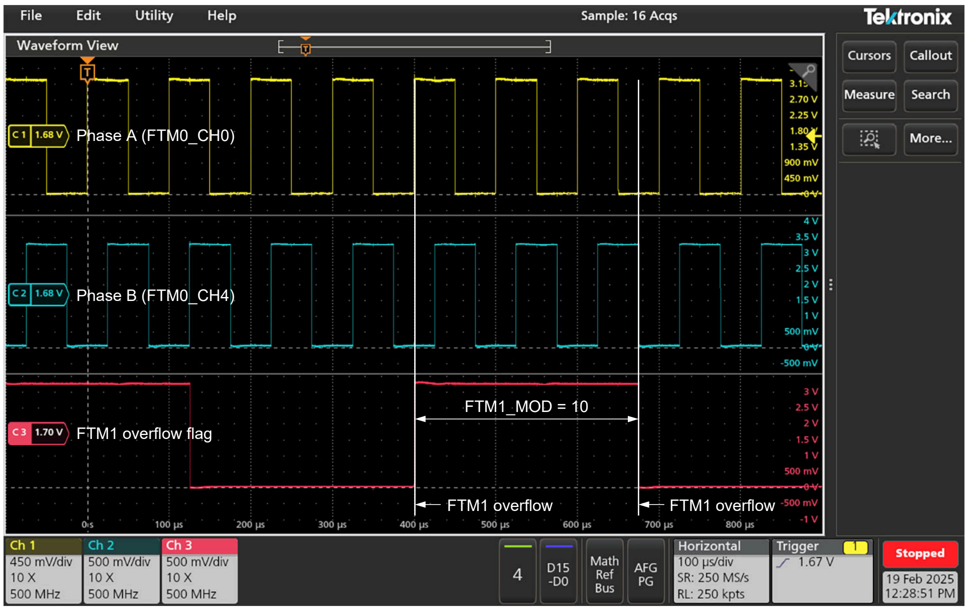


Figure 10. Quadrature decoder mode

Figure 10 shows the Quadrature decoder mode signal output:

- The scope channels CH1 and CH2 display the encoder signals Phase A and Phase B that are input to the FTM1 module (generated from `FTM0_CH0` and `FTM0_CH4`). The scope channel CH3 represents the FTM1 counter overflow interrupt generated every time the FTM counter reaches the value of the MODULO register (every time 10 rising/falling edges of the Phase A and Phase B signals are detected).
- When the FTM1 timer overflow occurs, the TOF bit and TOFDIR are set because the Phase B signal lags the Phase A signal.

4 FTM features

4.1 Masking, inverting, and software-controlling features

The masking and software-controlling features force the FTM channels' output to the inactive/required state, keeping a constant logic to the channels' output. These FTM functions can turn the power devices on/off via software. The inverting feature swaps the signals between the FTM channel pairs.

The masking feature is used in BLDC motor-control applications. A 3-phase BLDC motor is normally supplied by a 3-phase power inverter equipped with either six MOSFETs or six IGBTs (Insulated-Gate Bipolar Transistors). A commonly used control technique for BLDC motors is the 6-step commutation method. In this control technique, two phases are energized and the third one is disconnected. Therefore, both power devices of the corresponding inverter branch must be turned off. The masking feature of the FTM module can be implemented in this application to fulfill such function. Each FTM channel has a corresponding bit in the `FTM_OUTMASK` register. Any write to the `OUTMASK` register just latches the value in its write buffer. The output logic of the FTM channels can then be updated immediately after writing into the `FTM_OUTMASK` register while the FTM clock is disabled (at each rising edge of the FTM input clock when FTM_SYNC[SYNCHOM] = 0) or updated by the software or hardware synchronization, enabling the SWOM bit or the HWOM bit in the FTM_SYNCONF register (see Section 4.3). When the `FTM_CHx` channel is masked, its output logic is kept high/low, according to the bit enabled in the `FTM_POL` register.

The software-controlling feature forces the output of the FTM channels to the software-defined values by setting or clearing the `CHxOC` bits in the FTM_SWOCTRL register. If the `CHxOC` bits are cleared, the output of the FTM channels remains driven by the PWM or another source. If the `CHxOC` bits are set, the output of the corresponding FTM channel is affected by the software. The `CHxOCV` bit in the `FTM_SWOCTRL` register controls the logic of the individual FTM channel. The output of the `FTM_CHx` channel is high when CHxOCV = 1. Otherwise, it is 0.

The inverting function is commonly used in DC motor-control applications. The DC motor is supplied by the H-bridge and one channel pair (`FTM0_CH0`/`CH1`, for example) is connected to one branch while the other channel pair (`FTM0_CH2`/`CH3`, for example) is connected to another branch. Both channel pairs work in the complementary mode. Such an application requires that the diagonal power devices turn on/off at the same time, which means that the `FTM0_CH0` and `FTM0_CH3` must have the same timing. To guarantee such a condition, the `FTM0_CH1` and `FTM0_CH3` must work in the inverting mode. The `INVmEN` bit in the `FTM_INVCTRL` register enables inverting the corresponding channel pair. The value written in the `FTM_INVCTRL` register can be updated from the buffer in the same way as the `FTM_OUTMASK` register.

The masking function is demonstrated in Section 4.3.3.

4.2 Fault control feature

The fault control of the FTM module plays an important role in motor-control applications because it provides an opportunity to protect 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 a case, 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 pin/pins. An interrupt can be generated after receiving the fault signal and the undesirable behavior can be mitigated.

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

FTM has multiple channels. However, FTM can’t 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 FTM channel can be disabled by the fault signal or not depends on the `FAULTENx` bit in the `FTM_COMBINE` register. The resulting fault signal can disable all FTM channels or disable only the even channels (`FTM_CH0`/`CH2`/`CH4`/`CH6`). The selection depends on the FAULTM bit field in the `FTM_MODE` register.

There are two ways of recovering the output of the PWM channels: the automatic fault clearing set by FAULTM[1:0] = 1:1 and the manual fault clearing when FAULTM[1:0] = 0:1 or 1:0. The PWM signals are recovered from the safety state in the next PWM cycle without any software intervention when the automatic fault-clearing mode is selected. To recover the PWM signals in the manual clearing mode, clear the `FAULTF` bit in the `FTM_MODE` register and then enable the PWM in the next PWM cycle.

In the following code example, initiate the two FTM modules to demonstrate the fault-control feature. While `FTM0` is configured to generate the CPWM, `FTM1` is used to generate the fault signal that affects the output of the `FTM0` channels externally through the fault pin `FTM0_FLT3`.

```
void CPWM_and_Fault_Control()
  {
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
  PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
  PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
  PORTD->PCR[4]  = PORT_PCR_MUX(2); // FTM0_FLT3
  /* Enable combine, complementary mode and dead-time for pair channel 0/1 and 4/5 */
  FTM0->COMBINE = FTM_COMBINE_COMBINE0_MASK|FTM_COMBINE_COMP0_MASK|FTM_COMBINE_DTEN0_MASK
              | FTM_COMBINE_FAULTEN0_MASK|FTM_COMBINE_COMBINE2_MASK|FTM_COMBINE_COMP2_MASK
              | FTM_COMBINE_DTEN2_MASK|FTM_COMBINE_FAULTEN2_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(4800-1);
  /* 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[4].CnSC = FTM_CnSC_ELSB_MASK;
  FTM0->CONTROLS[5].CnSC = FTM_CnSC_ELSB_MASK;
  /* Set Channel Value */
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200);
  FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(3600);
  FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(2400);
  FTM0->CONTROLS[5].CnV=FTM_CnV_VAL(4800);
  /* FTM counter reset */
  FTM0->CNT = 0;
  /* Insert deadtime (1us) */
  FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) |
  FTM_DEADTIME_DTVAL(3);
  /* Set PTD4 pin as a fault input FTM0_FLT3 */
  FTM0->FLTCTRL = FTM_FLTCTRL_FAULT3EN_MASK;
  /* Enable fault control for all channels and select automatic fault clearing mode */
  FTM0->MODE |= FTM_MODE_FAULTM(3);
  /* Safe value is set as a low after fault input is detected */
  FTM0->POL = (~FTM_POL_POL0_MASK) & (~FTM_POL_POL1_MASK) & (~FTM_POL_POL4_MASK) &
 (~FTM_POL_POL5_MASK);
  /* A 1 at the fault input indicates the fault */
  FTM0->FLTPOL &= ~FTM_FLTPOL_FLT3POL_MASK;
  /* Select clock and enable PWM */
  FTM0->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK
           | FTM_SC_PWMEN5_MASK;
}

void FTM1_PWM_Output()
  {
  /* Enable clock for PORTA */
  CLOCK_EnableClock(kCLOCK_PortA);
  /* Enable clock for FTM1 */
  CLOCK_EnableClock(kCLOCK_Ftm1);
  /* Enable registers updating from write buffers */
  FTM1->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM1 */
  PORTA->PCR[16] = PORT_PCR_MUX(2); // FTM1, Channel3

  /* Set Modulo in initialization stage (1.2kHz PWM frequency @48MHz system clock) */
  FTM1->MOD = FTM_MOD_MOD(40000-1);
  /* Set CNTIN in initialization stage */
  FTM1->CNTIN = FTM_CNTIN_INIT(0);
  /* Enable high-true pulses of PWM signals */
  FTM1->CONTROLS[3].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
  /* Set channel value in initialization stage */
  FTM1->CONTROLS[3].CnV=FTM_CnV_VAL(20000); // 50% duty cycle

  /* Reset counter */
  FTM1->CNT = 0;
  /* Select clock */
  FTM1->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN3_MASK;
}
```

The resulting behavior of the code example is as shown in Figure 11.

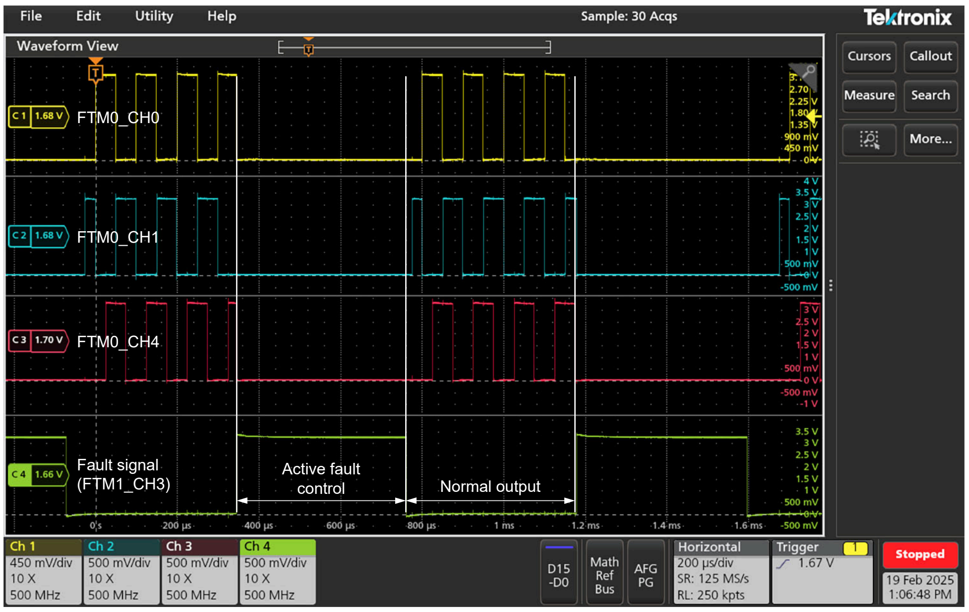


Figure 11. Fault control of CPWM

Figure 11 shows the fault control of CPWM signal output:

- The scope channels CH1, CH2, and CH3 represent the `FTM0` module channels `FTM0_CH0`, `FTM0_CH1`, and `FTM0_CH4`, respectively. Channel CH4 shows the external fault signal (generated from `FTM1_CH3`) whose frequency is about one eighth of the FTM0 frequency.
- `FTM0_CH0` and `FTM0_CH1` work in the complementary and combine mode, same as `FTM0_CH4` and `FTM0_CH5`.

- When a rising edge is detected on the `FTM0_FLT3` fault input pin (generated from `FTM1_CH3`), all FTM0 channels are forced to the safe (low) logic. Figure 11 also shows that all FTM0 channels are recovered at the next counter cycle, when the fault input signal is 0.

4.3 Updating FTM registers

In motor-control or switched-mode power supply applications, the PWM duty cycle controls the voltage applied on the load. To control the PWM duty cycle, set the appropriate values in the `FTM_CnV` and `FTM_CNTIN` registers. Some applications require to change the period of the PWM as well. The period is controlled by the value in the `FTM_MOD` register.

The `FTM_CnV`, `FTM_CNTIN`, and `FTM_MOD` are double-buffered registers which means that a write to these registers just latches their values into their write buffers. The double-buffered registers of the FTM module are the Channel (n) Value (`FTM_CnV`), Counter Initial Value (`FTM_CNTIN`), Modulo (`FTM_MOD`), Half Cycle Register (`FTM_HCR`), Output Mask (`OUTMASK`), Software Output Control (`SWOCTRL`), and Inverting Control register (`INVCTRL`). The double-buffered registers can be updated with their buffered values according to the updating scheme chosen. The current implementation on the MCX E24x device enables using the half or full cycle reload strategy, software or hardware PWM synchronization, or it is possible to bypass the buffers. Table 3 summarizes all double-buffered registers and their updating methods.

Table 3. Updating double-buffered FTM register

Update technique	Double-buffered FTM registers	Note
Initialization stage CLKS[1:0] = 0:0	CnV, CNTIN, MOD, HCR	The registers are loaded immediately.
TPM compatibility CLKS[1:0] ≠ 0:0, FTMEN = 0	CnV, CNTIN, MOD, HCR	The CNTIN register is loaded immediately and the CnV, MOD, and HCR registers at the end of the FTM counter period. The CnV register is loaded immediately in the output compare mode.
Software synchronization CLKS[1:0] ≠ 0:0, FTMEN=1, SYNCMODE = 1	CnV, CNTIN, MOD, HCR (SWWRBUF = 1)	The registers are updated by the software trigger-enabling SWSYNC bit. If SWRSTCNT = 1, the software trigger resets the FTM counter reset.
	SWOCTRL (SWOC = 1, SWSOC = 1)	The registers are updated by the software trigger-enabling SWSYNC bit.
	OUTMASK (SYNCHOM = 1, SWOM = 1)	
	INVCTRL (INVC = 1, SWINVC = 1)	
Hardware synchronization CLKS[1:0] ≠ 0:0, FTMEN=1, SYNCMODE = 1	CnV, CNTIN, MOD, HCR (HWWRBUF = 1)	The registers are updated when TRIGn = 1 and the hardware trigger is detected at the trigger (n) input signal. If HWTRIGMODE = 1, the hardware trigger clears the TRIGn bit.
	SWOCTRL (SWOC = 1, HWSOC = 1)	
	OUTMASK (SYNCHOM = 1, HWOM = 1)	
	INVCTRL (INVC = 1, HWINVC = 1)	
Half and full cycle reload strategy	CnV, CNTIN, MOD, HCR	In the up-counting mode, the synchronization points are when the FTM counter changes from MOD to CNTIN, enable CNTMIN, or CNTMAX bit.

Table 3. Updating double-buffered FTM register*...continued*

Update technique	Double-buffered FTM registers	Note
CLKS[1:0] ≠ 0:0, FTMEN=1		For the up/down-counting mode, see Table 4.

Table 4. Reload opportunities in up/down-counting mode

FTM_SYNC bits	Reload opportunities selected
CNTMIN = 0 and CNTMAX = 0	When the counter turns from up to down (compatibility mode).
CNTMIN = 1 and CNTMAX = 0	When the counter turns from down to up.
CNTMIN = 0 and CNTMAX = 1	When the counter turns from up to down.
CNTMIN = 1 and CNTMAX = 1	When the counter turns from down to up and when the counter turns from up to down.

The configuration steps of the updating methods are discussed in detail in the following sections.

4.3.1 Updating FTM registers in initialization stage

In the initialization stage, when the FTM module is disabled (CLKS[1:0] = 0:0), the FTM registers are updated immediately after writing to the `FTM_CnV`, `FTM_CNTIN`, and `FTM_MOD` registers. After the initialization step, the FTM module starts to generate the PWM by setting CLKS[1:0] ≠ 0:0 and enabling the `PWMENn` bits in the `FTM_SC` register.

4.3.2 Half- and full-cycle reload strategies

To change the PWM duty cycle or the PWM time period while the FTM counter is running, it is possible to apply the half- and full-cycle reload strategies. 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 EPWM, the half and full cycle reload opportunities can update the FTM registers according to these steps:

1. Initialize FTM0 to generate the EPWM and enable the FTMEN bit in the `FTM_MODE` register and the RIE bit in the `FTM_SC` register.
2. Enable the `HCSEL` bit in the `FTM_PWMLOAD` register and adjust the value in the `FTM_HCR` register to MOD/2 for the half-cycle reload opportunity and HCSEL = 0 for the full-cycle reload opportunity.
3. 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.

A different scenario must be considered when the up/down-counting mode is selected to generate the CPWM:

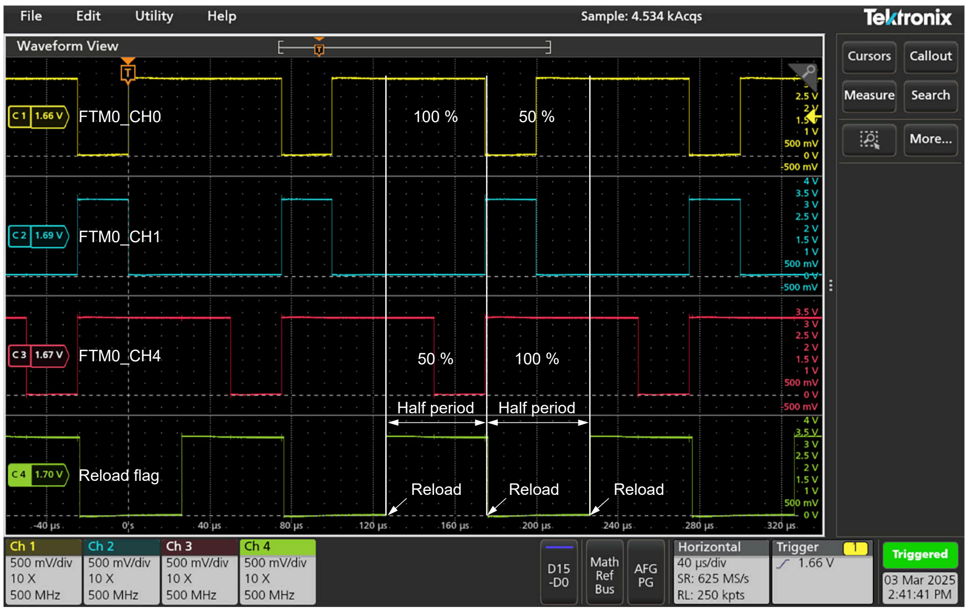
1. Initialize FTM0 to generate the CPWM and enable the FTMEN bit in the `FTM_MODE` register and the RIE bit in the `FTM_SC` register.
2. Enable the CNTMIN and CNTMAX bits in the `FTM_SYNC` register for the half-cycle reload opportunity and the CNTMIN = CNTMAX = 0 for the full-cycle reload opportunity.
3. 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.

The following code example shows a part of the `FTM0` initialization for the CPWM and the half-cycle reload opportunity interrupt routine, where the values `C0V` and `C2V` of the channels are modified according to the variable flag.

```
void Half_and_Full_Cycle_Reload()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
  PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
  PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
  /* Select up-down counter for Center-Aligned PWM */
  FTM0->SC = FTM_SC_CPWMS_MASK;
  /* Complementary mode, sync and dead-time enable for channel0 and channel1 */
  FTM0->COMBINE = FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK;
  /* Complementary mode, sync and dead-time enable for channel4 and channel5 */
  FTM0->COMBINE |= FTM_COMBINE_SYNCEN2_MASK | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_DTEN2_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(2400-1);
  /* 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[4].CnSC = FTM_CnSC_ELSB_MASK;
  FTM0->CONTROLS[5].CnSC = FTM_CnSC_ELSB_MASK;
  /* Set Channel Value */
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2400);
  FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(1200);
  /* FTM counter reset */
  FTM0->CNT = 0;

  /* Enable clock for PORTE */
  CLOCK_EnableClock(kCLOCK_PortE);
  PORTE->PCR[7] = PORT_PCR_MUX(1); // PTE7 set as reload flag
  /* GPIO Initialize */
  gpio_pin_config_t gpio_config =
  {
    kGPIO_DigitalOutput,
    0
  };
  GPIO_PinInit(GPIOE, 7, &gpio_config);
  EnableIRQ(FTM0_Ovf_Reload_IRQn);

  if(g_hf_reload) // half cycle reload enable flag
  {
    /* Enable half cycle reload */
    FTM0->SYNC = FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK;
  }
  // else: full cycle reload enable
  /* Select clock, enable reload opportunity interrupt and enable PWM generation */
  FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK
            | FTM_SC_PWMEN5_MASK | FTM_SC_RIE_MASK;
}

void FTM0_Ovf_Reload_IRQHandler()
{
  FTM0->PWMLOAD |= FTM_PWMLOAD_LDOK_MASK; // Set LDOK bit
  if(g_reload_flag)  // reload flag
  {
    FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2400); // 50%
    FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(1200); // 25%
  }
  else
  {
    FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // 25%
```

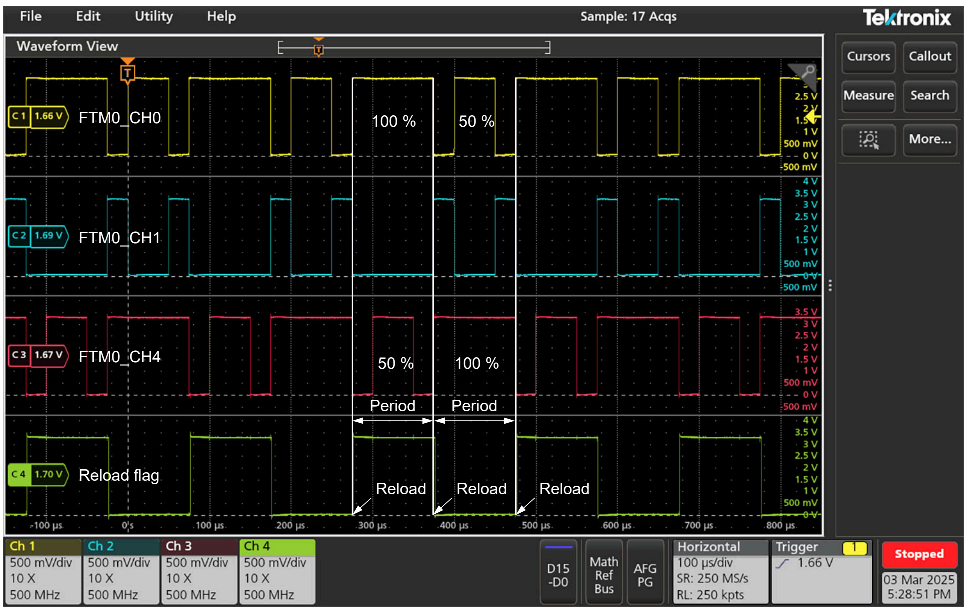
```
      FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(2400); // 50%
    }
    g_reload_flag = !g_reload_flag;
    GPIO_PortToggle(GPIOE, 1 << 7);
    FTM0->SC &= ~FTM_SC_RF_MASK; // Clear Timer Flag
  }
```

Figure 12 and Figure 13 demonstrate the half- and full-cycle reload strategy of the FTM registers performing the CPWM.



aaa-060036

Figure 12. CPWM and half-cycle reload strategy



aaa-060037

Figure 13. CPWM half and full-cycle reload strategy

Figure 12 and Figure 13 show the CPWM half- and full-cycle reload signal output:

- The scope channels CH1, CH2, and CH3 represent the `FTM0_CH0`, `FTM0_CH1`, and `FTM0_CH4` of the `FTM0` module, respectively. CH4 is used to show the reload opportunity of the `FTM0` registers that occurs every half/ full-cycle of the up-down counter.
- The `FTM0_CH0` and `FTM0_CH1` work in the complementary mode, same as the `FTM0_CH4` and `FTM0_CH5`.
- In the first reload opportunity, the 100 % and 50 % duty cycles are applied to the complementary channel pairs `FTM0_CH0/CH1` and `FTM0_CH4/CH5`, respectively. At the next reload opportunity, the `C0V` and `C4V` registers are updated from their new buffed values, therefore the duty cycles of `FTM0_CH0/CH1` and `FTM0_CH4/CH5` are changed to 50 % and 100 %, respectively.

4.3.3 Updating FTM registers by software trigger - Software synchronization

All double-buffered FTM registers can be updated by the software synchronization, enabling the `SWSYNC` bit in the FTM_SYNC register (as shown in Table 3). The update can take place immediately or at the next loading point selected, according to the boundary cycle and the loading points of the selected counting mode.

In the up-counting mode, the boundary cycle is defined as when the counter wraps to its initial value (CNTIN). When in the up/down-counting mode, the boundary cycle is defined as when the counter turns from the down-counting to the up-counting and from the up-counting to the down-counting. In the up-counting mode, the loading points are enabled if one of the `CNTMIN` or `CTMAX` bits is **1**.

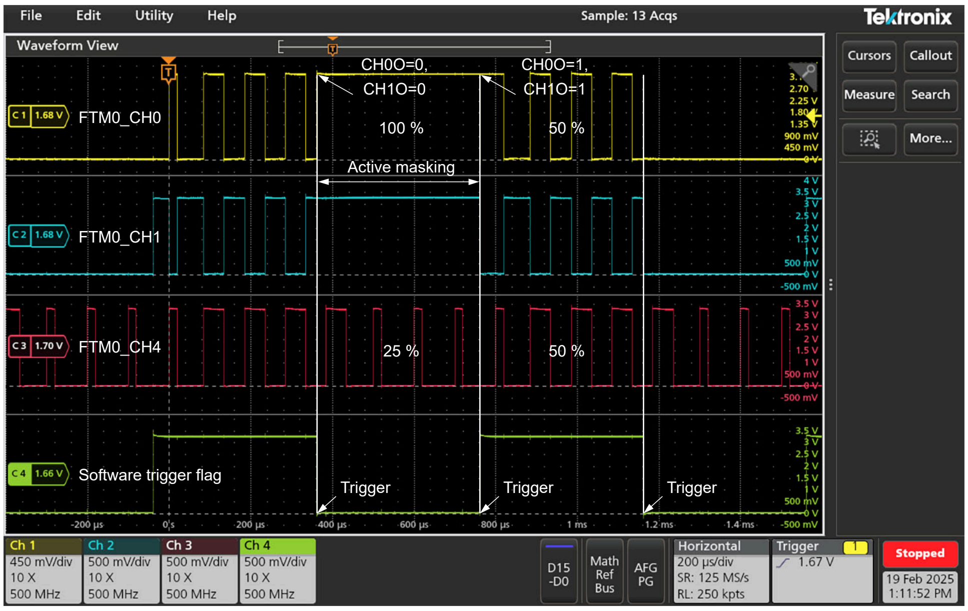
If the SWRSTCNT bit is set in the `FTM_SYNCONF` register, the FTM counter restarts from the `FTM_CNTIN` and the FTM registers are updated immediately. If the SWRSTCNT bit is cleared, the FTM counter remains counting and the FTM registers are updated at the next loading point.

Perform the following steps to configure the FTM module to generate the CPWM and use the software synchronization to update the `FTM_CnV` and `FTM_OUTMASK` registers:

1. Initialize FTM0 to generate the CPWM and enable the `FTMEN` bit in the `FTM_MODE` register and the RIE bit in the `FTM_SC` register.
2. Enable the `SYNCMODE`, `SWWRBUF`, and `SWOM` bits in the `SYNCONF` register to select the software synchronization of the `CnV` and `OUTMASK` registers.
3. Enable the software trigger by FTM_SYNC[SWSYNC] = 1 and wait for the next loading point.
4. In the reload opportunity interrupt routine, change the value of the `CnV` and `OUTMASK` registers and enable the software trigger by SWSYNC.

The following code example shows a part of the FTM0 initialization for the CPWM and demonstrates the software synchronization for updating the `FTM_CnV` and `FTM_OUTMASK` registers.

```
void FTM0_CPWM_SoftSync()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
  PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
  PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
  /* Select up-down counter for Center-Align PWM */
  FTM0->SC = FTM_SC_CPWMS_MASK;
  /* Complementary mode, sync and dead-time enable for channel0 and channel1 */
  FTM0->COMBINE |= FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK;
  /* Complementary mode, sync and dead-time enable for channel4 and channel5 */
  FTM0->COMBINE |= FTM_COMBINE_SYNCEN2_MASK | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_DTEN2_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(2400-1);
  /* 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[4].CnSC = FTM_CnSC_ELSB_MASK;
  FTM0->CONTROLS[5].CnSC = FTM_CnSC_ELSB_MASK;
  /* Set Channel Value */
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // 50%
  FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(600); // 25%
  /* FTM counter reset */
  FTM0->CNT = 0;

  /* Enable clock for PORTE */
  CLOCK_EnableClock(kCLOCK_PortE);
  PORTE->PCR[7] = PORT_PCR_MUX(1); // PTE7 set as reload flag

  /* GPIO Initialize */
  gpio_pin_config_t gpio_config =
  {
  kGPIO_DigitalOutput,
  0
  };
  GPIO_PinInit(GPIOE, 7, &gpio_config); // GPIO init
  EnableIRQ(FTM0_Ovf_Reload_IRQn);      // Enable interrupt

  /* Select clock, enable reload opportunity interrupt */
  FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_RIE_MASK;
  /* Enable reload opportunity interrupt when FTM counter reach CNTMAX value */
  FTM0->SYNC |= FTM_SYNC_SYNCHOM_MASK | FTM_SYNC_CNTMAX_MASK;
  /* Allow each fourth reload opportunity interrupt */
  FTM0->CONF = FTM_CONF_LDFQ(3);
  /* Enable software synchronization */
  FTM0->SYNCONF = FTM_SYNCONF_SYNCMODE_MASK | FTM_SYNCONF_SWWRBUF_MASK | FTM_SYNCONF_SWOM_MASK;
  /* Enable PWM generation */
  FTM0->SC |= FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK | FTM_SC_PWMEN5_MASK;
}

void FTM0_Ovf_Reload_IRQHandler()
{
  static bool flag = false;

  if(g_reload_flag)
  {
   /* Enable FTM0_CH0/FTM0_CH1 mask */
   FTM0->OUTMASK = FTM_OUTMASK_CH0OM_MASK | FTM_OUTMASK_CH1OM_MASK;
   FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(600); // 25%
   FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(600); // 25%
   if(flag)
   {
    /* Set FTM0_CH0/FTM0_CH1 output polarity to high */
    FTM0->POL = FTM_POL_POL0_MASK | FTM_POL_POL1_MASK;
   }
   else
   {
    /* Set FTM0_CH0/FTM0_CH1 output polarity to low */
    FTM0->POL &= (~FTM_POL_POL0_MASK) & (~FTM_POL_POL1_MASK);
   }
   flag = !flag;
  }
  else
  {
  /* Set FTM0_CH0/FTM0_CH1 output polarity to low */
   FTM0->POL &= (~FTM_POL_POL0_MASK) & (~FTM_POL_POL1_MASK);
   /* Disable FTM0_CH0/FTM0_CH1 mask */
   FTM0->OUTMASK &= (~FTM_OUTMASK_CH0OM_MASK) & (~FTM_OUTMASK_CH1OM_MASK);
   FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // 50%
   FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(1200); // 50%
  }
  g_reload_flag = !g_reload_flag;
  GPIO_PortToggle(GPIOE, 1 << 7); // Toggle PTE7 to show reload opportunities
  FTM0->SYNC |= FTM_SYNC_SWSYNC_MASK; // Software sync
  FTM0->SC &= ~FTM_SC_RF_MASK; // Clear Timer Flag
}
```



aaa-060038

Figure 14. Software synchronization of CPWM

Figure 14 shows the software synchronization of CPWM signal output:

- The scope channels CH1, CH2, and CH3 represent `FTM0_CH0`, `FTM0_CH1`, and `FTM0_CH4` of the FTM0 module, respectively. Channel CH4 is used to show the software trigger which occurs every fourth PWM period.
- `FTM0_CH0` and `FTM0_CH1` work in the complementary mode, same as `FTM0_CH4` and `FTM0_CH5`.
- The duty cycles of channel pairs `FTM0_CH0/FTM0_CH1` and `FTM0_CH4/FTM0_CH5` change from 50 % to 25 % (and vice-versa), depending on the software trigger events that update the C0V and C4V registers. Moreover, the first complementary channel pair `FTM0_CH0/FTM0_CH1` shows the masking feature.

4.3.4 Updating FTM registers by hardware trigger - Hardware synchronization

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

Three hardware trigger signal inputs of the FTM module can be selected depending on the enabled `TRIGn` bit in the `FTM_SYNC` register. If `TRIG0` is set, the hardware trigger source comes from many other modules, such as LPIT and CMP (and other) through a flexible TRIGMUX module. By setting the TRIG1 bit in the `FTM_SYNC` register, a hardware trigger is generated according to the `FTM0SYNCBIT` bit in the `SIM_FTMOPT1` register. In this section, the third option is demonstrated by enabling the `TRIG2` bit. In this case, the `FTMx_FLT0` fault pin is chosen as the hardware trigger input for the FTM hardware synchronization.

1. Initialize `FTM0` to generate the CPWM and enable the `FTMEN` bit in the `FTM_MODE` register.
2. Enable the `SYNCMODE` and `HWWRBUF` bits in the `FTM0_SYNCONF` register to select the hardware synchronization.
3. Enable the TRIG2 bit in the `FTM0_SYNC` register to select the `FTM0_FLT0` fault pin as the hardware trigger input for the `FTM0` hardware synchronization.
4. Initialize the `FTM1` module to generate a periodic signal for the `FTM0` hardware trigger input. To better demonstrate the `FTM0` hardware synchronization, the frequency of `FTM1` must be lower than the frequency of `FTM0`.

5. Generate a reload opportunity interrupt by enabling the `RIE` bit in the `FTM1_SC` register and change the values in the C0V and C4V registers. Wait for a hardware trigger event that updates the values from the buffers.

The core code for hardware synchronization is as follows:

```
void FTM0_CPWM_HardSync()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for PORTA */
  CLOCK_EnableClock(kCLOCK_PortA);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1
  PORTB->PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel4
  PORTB->PCR[17] = PORT_PCR_MUX(2); // FTM0, Channel5
  PORTA->PCR[14] = PORT_PCR_MUX(2); // FTM0_FLT0
  /* Set PTA14 pin as a fault input FTM0_FLT0 */
  FTM0->FLTCTRL = FTM_FLTCTRL_FAULT0EN_MASK;
  /* Select up-down counter for Center-Align PWM */
  FTM0->SC = FTM_SC_CPWMS_MASK;
  /* Complementary mode, sync and dead-time enable for channel0 and channel1 */
  FTM0->COMBINE |= FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK;
  /* Complementary mode, sync and dead-time enable for channel4 and channel5 */
  FTM0->COMBINE |= FTM_COMBINE_SYNCEN2_MASK | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_DTEN2_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(2400-1);
  /* 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[4].CnSC = FTM_CnSC_ELSB_MASK;
  FTM0->CONTROLS[5].CnSC = FTM_CnSC_ELSB_MASK;
  /* Set Channel Value */
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // 50%
  FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(600); // 25%
  /* FTM counter reset */
  FTM0->CNT = 0;

  /* Select and enable clock */
  FTM0->SC |= FTM_SC_CLKS(1);
  /* Enable hardware synchronization */
  FTM0->SYNCONF = FTM_SYNCONF_SYNCMODE_MASK | FTM_SYNCONF_HWWRBUF_MASK;
  /* Enable FTM counter reset after hardware trigger */
  FTM0->SYNCONF |= FTM_SYNCONF_HWRSTCNT_MASK | FTM_SYNCONF_HWTRIGMODE_MASK;
  /* Select FTM0_FLT0 pin as a hardware trigger input */
  FTM0->SYNC = FTM_SYNC_TRIG2_MASK;
  /* Enable PWM generation */
  FTM0->SC |= FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN4_MASK | FTM_SC_PWMEN5_MASK;
}

void FTM1_CPWM_SyncOut()
{
  /* Enable clock for PORTA */
  CLOCK_EnableClock(kCLOCK_PortA);
  /* Enable clock for FTM1 */
  CLOCK_EnableClock(kCLOCK_Ftm1);
  /* Enable registers updating from write buffers */
  FTM1->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM1 */
  PORTA->PCR[16] = PORT_PCR_MUX(2); // FTM1, Channel3

  FTM1->SC=FTM_SC_CPWMS_MASK; // Select up-down counter for Center-Align PWM
  /* Set Modulo in initialization stage (2kHz PWM frequency @48MHz system clock) */
  FTM1->MOD = FTM_MOD_MOD(12000-1);
  /* Set CNTIN in initialization stage */
  FTM1->CNTIN = FTM_CNTIN_INIT(0);
  /* Enable high-true pulses of PWM signals */
  FTM1->CONTROLS[3].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
  /* Set channel value in initialization stage */
  FTM1->CONTROLS[3].CnV=FTM_CnV_VAL(6000); // 50% duty cycle

  /* Reset counter */
  FTM1->CNT = 0;
  /* Select clock */
  FTM1->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN3_MASK | FTM_SC_RIE_MASK;
  FTM1->SYNC = FTM_SYNC_CNTMAX_MASK;

  EnableIRQ(FTM1_Ovf_Reload_IRQn); // Enable interrupt
}

void FTM1_Ovf_Reload_IRQHandler()
{
  if(g_reload_flag)
  {
   FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1200); // 50%
   FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(600);  // 25%
  }
  else
  {
   FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(600); // 25%
   FTM0->CONTROLS[4].CnV=FTM_CnV_VAL(1200); // 50%
  }
  g_reload_flag = !g_reload_flag;
  FTM1->SC &= ~FTM_SC_RF_MASK; // Clear Reload Flag bit
}
```

Figure 15 shows the resulting behavior of the code example.

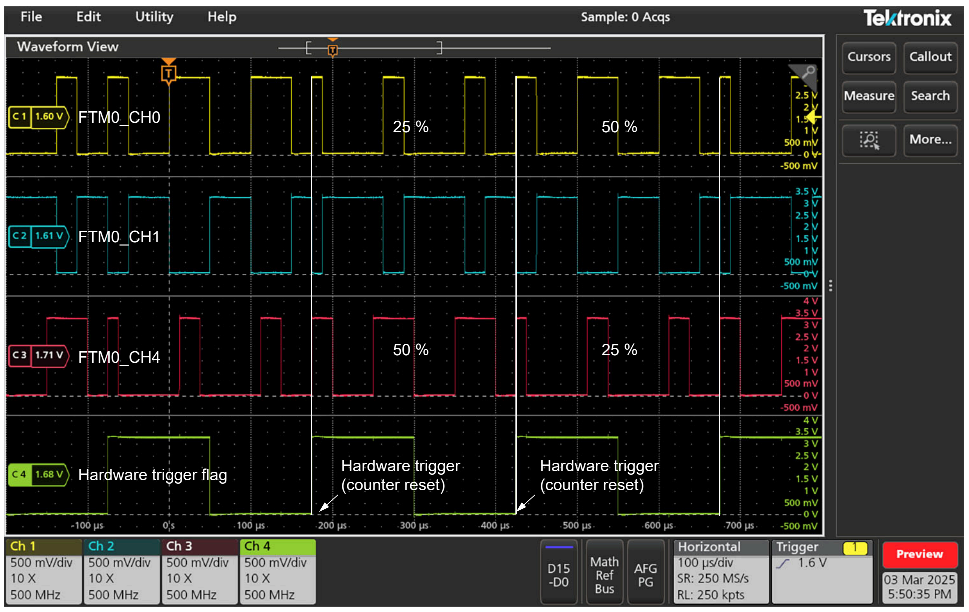

File
Edit
Utility
Help
Sample: 0 Acqs
Tektronix
Waveform View
Cursors
Callout
Measure
Search
More...
FTM0_CH0
25 %
50 %
FTM0_CH1
FTM0_CH4
50 %
25 %
Hardware trigger flag
Hardware trigger (counter reset)
Hardware trigger (counter reset)
Ch 1 500 mV/div 10 X 500 MHz
Ch 2 500 mV/div 10 X 500 MHz
Ch 3 500 mV/div 10 X 500 MHz
Ch 4 500 mV/div 10 X 500 MHz
Horizontal 100 µs/div SR: 250 MS/s RL: 250 kpts
Trigger 1.6 V
Preview
03 Mar 2025 5:50:35 PM
aaa-060039

Figure 15. Hardware synchronization of CPWM

Figure 15 shows the hardware synchronization of CPWM signal output.

- The scope channels CH1, CH2, and CH3 represent the FTM0 module channels `FTM0_CH0`, `FTM0_CH1`, and `FTM0_CH4`, respectively. Channel CH4 shows the hardware trigger generated from `FTM1_CH3`.
- `FTM0_CH0` and `FTM0_CH1` work in the complementary mode, same as `FTM0_CH4` and `FTM0_CH5`.
- After receiving the first hardware trigger, the duty cycles of the channel pair `FTM0_CH0`/`FTM0_CH1` and the channel pair `FTM0_CH4`/`FTM0_CH5` change from 25 % to 50 % and from 50 % to 25 %, respectively.

The next hardware trigger updates the new values of the `C0V` and `C4V` registers and the duty cycle of the corresponding channel pair changes back to the previous value.

4.4 Updating FTM registers

There are multiple FTMs on the chip, but the multiple FTM modules are independent. If the application requires more PWM channels than what one FTM can provide, 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.

The MCX E24x device provides the GTB mechanism to synchronize multiple FTMs. The GTB is a synchronous signal generated by the master FTM that launches the counter of all FTMs used. These two conditions must be met when using the GTB function: each FTM must have the same clock source and each FTM must start at the same time.

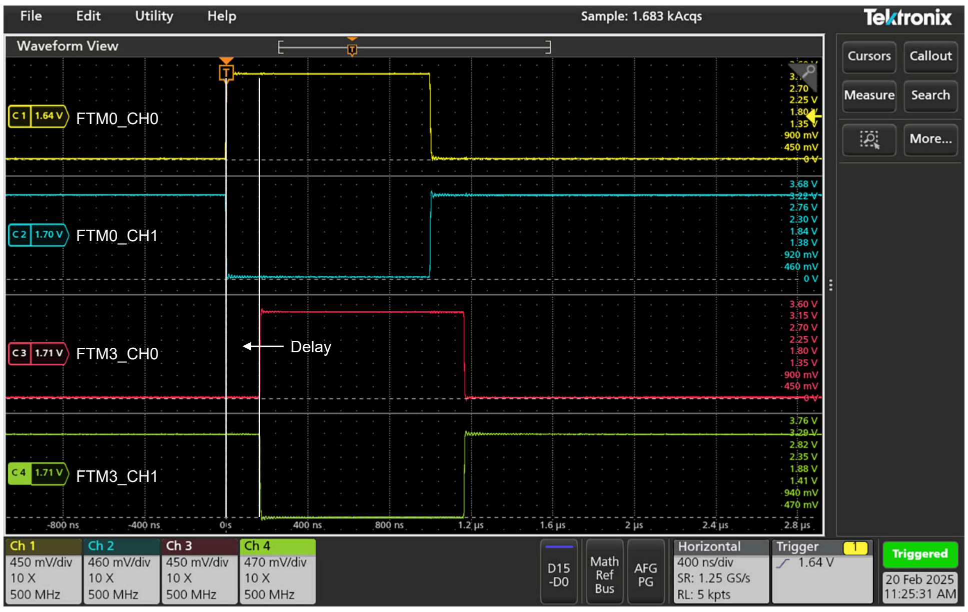
To enable the GTB feature, perform these steps for each participating FTM module:

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 following example shows the synchronization of two generating FMTs - FTM0 and FTM3. The duty cycles of both PWMs are adjusted to very low values to better demonstrate the functionality of the GTB mechanism.

```
void CPWM_and_Global_Time_Base()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable clock for FTM3 */
  CLOCK_EnableClock(kCLOCK_Ftm3);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  FTM3->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 & FTM3 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // Set PTD15 for FTM0 - Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // Set PTD16 for FTM0 - Channel1
  PORTB->PCR[8] = PORT_PCR_MUX(2); // Set PTB8 for FTM3 - Channel0
  PORTB->PCR[9] = PORT_PCR_MUX(2); // Set PTB9 for FTM3 - Channel1
  /* Select up-down counter for Center-Align PWM */
  FTM0->SC = FTM_SC_CPWMS_MASK;
  FTM3->SC = FTM_SC_CPWMS_MASK;
  /* Enable complementary mode for channels pair n=1 */
  FTM0->COMBINE = FTM_COMBINE_COMP0_MASK;
  FTM3->COMBINE = FTM_COMBINE_COMP0_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(2400-1);
  FTM3->MOD = FTM_MOD_MOD(2400-1);
  /* Set CNTIN */
  FTM0->CNTIN = FTM_CNTIN_INIT(0);
  FTM3->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;
  FTM3->CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;
  FTM3->CONTROLS[1].CnSC = FTM_CnSC_ELSB_MASK;
  /* Set Channel Value - 1% duty cycle */
  FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(24);
  FTM3->CONTROLS[0].CnV=FTM_CnV_VAL(24);
```

```
    /* FTM counter reset */
    FTM0->CNT = 0;
    FTM3->CNT = 0;
    /* Enable global time base to control FTM0 and FTM3 */
    FTM0->CONF = FTM_CONF_GTBEEN_MASK;
    FTM3->CONF = FTM_CONF_GTBEEN_MASK;
    /* Select clock */
    FTM0->SC |= FTM_SC_CLKS(1);
    FTM3->SC |= FTM_SC_CLKS(1);
    /* Synchronization signal for FTM0 and FTM3 */
    FTM0->CONF |= FTM_CONF_GTBEOUT_MASK;
    /* Enable PWM generation */
    FTM0->SC |= FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
    FTM3->SC |= FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
}
```



aaa-060040

Figure 16. CPWM with no GTB activated

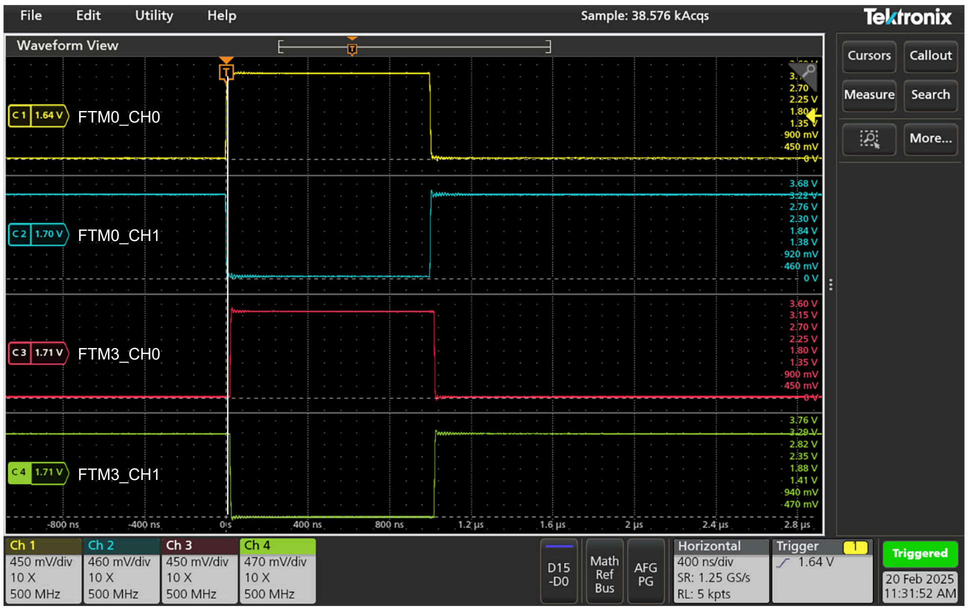

FTM0_CH0
FTM0_CH1
FTM3_CH0
FTM3_CH1
Ch 1 450 mV/div 10 X 500 MHz
Ch 2 460 mV/div 10 X 500 MHz
Ch 3 450 mV/div 10 X 500 MHz
Ch 4 470 mV/div 10 X 500 MHz
Horizontal 400 ns/div SR: 1.25 GS/s RL: 5 kpts
Trigger 1.64 V
Triggered
20 Feb 2025 11:31:52 AM
Tektronix
Sample: 38.576 kAcqs
aaa-060041

Figure 17. CPWM with GTB activated

Figure 16 and Figure 17 show the CPWM outputs of `FTM0` and `FTM3` with no GTB activated and with the GTB activated:

- The scope channels CH1, CH2, CH3, and CH4 represent the `FTM0_CH0`, `FTM0_CH1`, `FTM3_CH0`, and `FTM3_CH1`, respectively.
- `FTM0_CH0` and `FTM0_CH1` work in the complementary mode, same as `FTM3_CH0` and `FTM3_CH1`.
- The time base of the scope is adjusted so that a 1 % duty cycle of channels `FTM0` and `FTM3` can be clearly seen on the scope display. If the global time base is not activated, the up-down counters of the FTMs used are not perfectly aligned to each other, neither are their channel signals (see Figure 16). Such a problem is handled by applying the GTB function to the FTM modules (see Figure 17).

Note: *The GTB feature does not provide a continuous synchronization of the FTM counters, meaning that the FTM counters may lose the synchronization during the FTM operation. The GTB feature only enables the FTM counters to start their operation synchronously.*

4.5 ADC triggering by FTM and PDB modules

In motor-control applications and switched-mode power supply applications, it is necessary to measure the currents at the center of the PWM signal. The power devices do not switch at this instant time, so the voltage drop sensed on the shunt resistor is stable.

All FTM modules of the MCX E24x device have up to eight channels. Six channels are enough to generate the PWM signals necessary to control commonly used 3-phase motors. The remaining two channels of the FTM module can then be used to control the instant of the ADC conversions.

The triggering signal of the FTM module can be selected in the `FTM_EXTTRIG` register. All FTM channels can be used to trigger the ADC module. For example, if CH0 is set in the `FTM_EXTTRIG` register, the FTM counter counts until it reaches the value written in the `C0V` register. Currently, `CH0` generates a signal that triggers the ADC module on every FTM counter period. If the `INITTRIG` bit is set in the `FTM_EXTTRIG` register, the trigger signal is generated every time the FTM counter reaches the value of the CNTIN register.

The default and suggested hardware triggering scheme for the ADC module is through the Programmable Delay Block (PDB). The current implementation of the MCX E24x device is equipped with two ADC modules and two PDB modules that work in pairs. This means that `PDB0` is linked with `ADC0` and `PDB1` is linked with `ADC1`. Each PDB module can be triggered either by software or hardware. The FTM module is one of the trigger sources that can be enabled through the `TRGMUX` registers `TRGMUX_PDB0` and `TRGMUX_PDB1`.

To generate the CPWM and to trigger `ADC0` by `FTM0` through `PDB0`, perform these steps:

1. Initialize the FTM0 module to generate the CPWM and enable the `INITTRIGEN` bit in the `FTM0_EXTTRIG` register to generate a hardware trigger for the `ADC0` module.
2. Configure the TRGMUX module to trigger `ADC0` by `FTM0` through the PDB0 module by writing `0x16` to the `SEL0` bit field in the `TRGMUX_PDB0` register.
3. Delay the `FTM0` initialization trigger by the `PDB0` block to trigger `ADC0` at the center of the PWM cycle by writing appropriate values into the `PDB0_MOD` and `PDB0_CH0DLY0` registers.
4. Initialize the ADC0 module and enable the ADTRG bit in the `ADC0_SC2` register to trigger ADC0 by the FTM0 module.

The following code example demonstrates the configuration of the `FTM0`, `PDB0`, and `ADC0` modules, where `FTM0` is used to generate the CPWM and to trigger `ADC0` through the `PDB0` module.

```
void FTM0_Init_For_ADC_Triggering()
{
  /* Enable clock for PORTD */
  CLOCK_EnableClock(kCLOCK_PortD);
  /* Enable clock for PORTB */
  CLOCK_EnableClock(kCLOCK_PortB);
  /* Enable clock for PORTC */
  CLOCK_EnableClock(kCLOCK_PortC);
  /* Enable clock for FTM0 */
  CLOCK_EnableClock(kCLOCK_Ftm0);
  /* Enable registers updating from write buffers */
  FTM0->MODE = FTM_MODE_WPDIS_MASK | FTM_MODE_FTMEN_MASK;
  /* Set PORT pins for FTM0 */
  PORTD->PCR[15] = PORT_PCR_MUX(2); // Set PTD15 for FTM0 - Channel0
  PORTD->PCR[16] = PORT_PCR_MUX(2); // Set PTD16 for FTM0 - Channel1
  PORTC->PCR[7] = PORT_PCR_MUX(1); // Set PTC7 as GPIO, ADC interrupt flag
  PORTB->PCR[2] = PORT_PCR_MUX(1); // Set PTB2 as GPIO, PDB interrupt flag

  /* GPIO Initialize */
  gpio_pin_config_t gpio_config =
  {
   kGPIO_DigitalOutput,
   0
  };
  GPIO_PinInit(GPIOC, 7, &gpio_config); // PTC7 init
  GPIO_PinInit(GPIOB, 2, &gpio_config); // PTB2 init

  /* Select up-down counter for Center-Align PWM */
  FTM0->SC = FTM_SC_CPWMS_MASK;
  /* Enable sync, complementary mode and dead-time for channels pair CH0/CH1 */
  FTM0->COMBINE = FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK;
  /* Set Modulo (10kHz PWM frequency @48MHz system clock) */
  FTM0->MOD = FTM_MOD_MOD(2400-1);
  /* 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(1200); // 50% duty cycle
  /* FTM counter reset */
  FTM0->CNT = 0;
  /* Insert deadtime (1us) */
  FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) |
  FTM_DEADTIME_DTVAL(3);
  /* Enable trigger generation when FTM counter = CNTIN */
  FTM0->EXTTRIG = FTM_EXTTRIG_INITTRIGEN_MASK;
  /* Enable clock and PWM */
  FTM0->SC |= FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
}
```

```
void PDB0_Init_For_ADC_Triggering()
{
  /* Enable clock for PDB0 */
  CLOCK_EnableClock(kCLOCK_Pdb0);
  /* Enable interrupt PDB0 */
  EnableIRQ(PDB0_IRQn);
  PDB0->MOD = 4800; // Set Modulo
  PDB0->CH[0].C1 = PDB_C1_TOS(1) | PDB_C1_EN(1); // Select and enable Channel0
  PDB0->CH[0].DLY[0] = 2400; // Set delay
  PDB0->IDLY = 0; // Set interrupt delay
  PDB0->SC |= PDB_SC_PRESCALER(0) | PDB_SC_MULT(0); // Select clock prescaler and mult factor
  /* Select trigger input source and enable interrupt */
  PDB0->SC |= PDB_SC_TRGSEL(0) | PDB_SC_PDBIE_MASK;
  /* Enable PDB and update PDB registers */
  PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;
}

void TRGMUX_Init_For_ADC_Triggering()
{
  /* Set FTM as a trigger source for PDB0 */
  TRGMUX->TRGCFG[TRGMUX_PDB0_INDEX] = TRGMUX_TRGCFG_SEL0(kTRGMUX_SourceFtm0InitTrg);
}

void ADC0_Init_For_ADC_Triggering()
{
  /* Disable clock for ADC0 */
  CLOCK_DisableClock(kCLOCK_Adc0);
  /* Enable clock for ADC0 */
  CLOCK_EnableClock(kCLOCK_Adc0);
  /* Set divide ratio to 1 and select 8-bit conversion */
  ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(0) | ADC_CFG1_ADICLK(0);
  /* Select hardware trigger */
  ADC0->SC2 = ADC_SC2_ADTRG_MASK;
  /* Select channel 12 as an input and enable conversion complete interrupt */
  ADC0->SC1[0] = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH(12);
  /* Enable interrupt for ADC0 */
  EnableIRQ(ADC0_IRQn);
}

void ADC0_IRQHandler()
{
  GPIO_PortToggle(GPIOC, 1 << 7); // Toggle PTC7 to show ADC interrupt
  g_ADC_rseult = ADC0->R[0]; // Read ADC result register
}

void PDB0_IRQHandler()
{
  GPIO_PortToggle(GPIOB, 1 << 2); // Toggle PTB2 to show PDB interrupt
  PDB0->SC &= ~PDB_SC_PDBIF_MASK; // Clear PDB interrupt flag
}
```

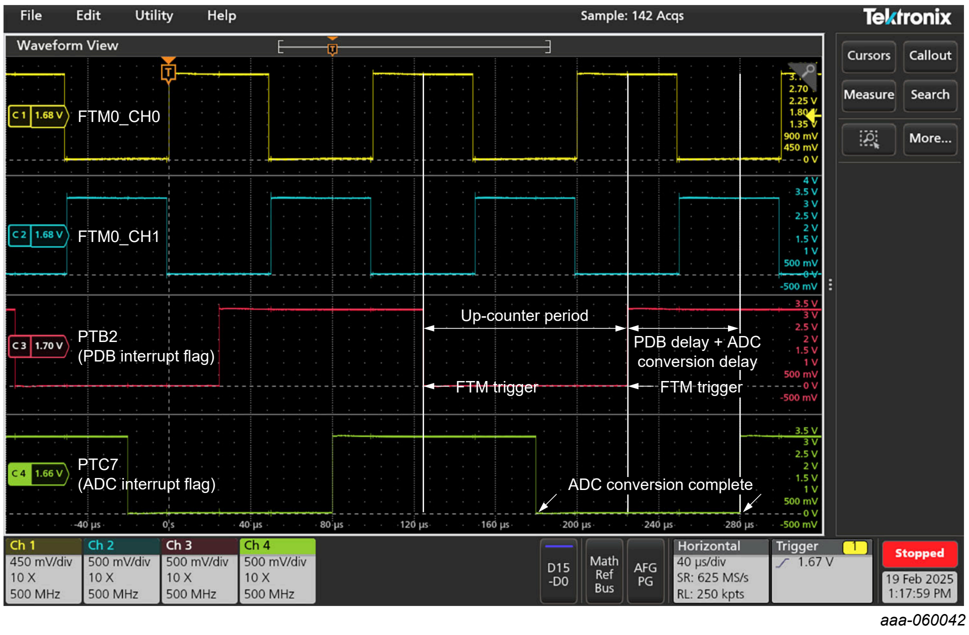


Figure 18. ADC triggering by FTM and PDB modules

Figure 18 shows the ADC triggering by FTM and PDB modules signal output:

- The scope channels CH1 and CH2 represent the FTM0 channels `FTM0_CH0` and `FTM0_CH1`, respectively. The scope channels CH3 and CH4 demonstrate the interconnection between the `FTM0`, `PDB0`, and `ADC0` modules. While channel CH3 represents the PTB2 pin toggling in the `PDB0` interrupt routine every time the `FTM0` counter reaches the value of the `CNTIN` register, channel CH4 displays the PTC7 pin toggling in the ADC0 interrupt routine every time the ADC0 conversion is completed (COCO = 1).
- The channel pair `FTM0_CH0`/`FTM0_CH1` works in the complementary mode.
- The `FTM0` trigger signal is delayed by the `PDB0` module, so that the ADC0 conversion occurs at the center of the PWM signal.

5 Note about the source code in the document

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

6 Revision history

Table 5 summarizes the revisions to this document.

Table 5. Revision history

Document ID	Release date	Description
AN14617 v1.0	15 July 2025	Initial public release

Legal information

Contents