NXP Semiconductors
Application Note

Document Number: AN5303
Rev. 0, 06/2016

S32K 上 FlexTimer 模块的特性和工作模式

URL：https://www.nxp.com/docs/en/application-note/AN5303.pdf

1. 简介

本文通过代码示例和示波器快照介绍了FlexTimer（FTM）模块的多种功能和操作模式。FTM模块在很多Kinetis和Vybird MCU上都有。但是，本文主要关注32位汽车MCU的S32K产品系列。FTM模块是定时器/PWM模块（TPM）的增强版本，可为电机控制、照明和开关电源应用提供新功能。主要的改进是：

- 计数器
- 硬件死区时间插入
- 故障控制输入
- 增强的触发功能
- 初始化和极性控制

屏蔽、反相、极性和错误控制以及硬件死区时间插入是专用于电机控制应用的FTM模块的主要功能。它们提供了更大的灵活性，并显著降低了CPU负载。另一方面，如果FTM模块不用于电机控制，则它保留标准定时器功能（例如输入捕获或输出比较模式）。这使得FTM更加灵活，涵盖了广泛的应用。

目录

2. FTM 概述

FTM模块有一个16位计数器，供FTM通道用于输入或输出模式。可以为FTM计数器选择三种可能的时钟源：系统时钟、固定频率时钟和外部时钟（有关详细的时钟说明，请参阅S32K系列参考手册中的时钟部分章节）。选定的计数器时钟信号然后通过一个预分频器，通过设置FTM_SC寄存器中的PS[2:0]将时钟源分频最大至128。如果FTM_SC寄存器中的CPWMS位被禁用，则FTM计数器以递增计数模式运行。否则，它将以加/减计数模式运行。

当前的S32K配备四个FTM。每个FTM模块有8个独立或成对工作的通道（CH0/CH1、CH2/CH3、CH4/CH5和CH6/CH7）。所有通道均可配置为输入捕获、输出比较或边沿/中心对齐PWM模式。在边沿对齐PWM模式下，FTM通道对可以在互补或/和组合模式下工作。此外，FTM1和FTM2能够在正交解码器模式下工作，从而能够处理来自增量旋转位置传感器（编码器）的信号。下表显示了通道的模式选择。

表1 . 通道模式选择

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

表2 . 双边沿捕获模式—边沿极性选择

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

在基于传感器的电机控制应用中，需要使用两个FTM模块。第一个FTM用于产生六个PWM信号来控制电机的加速度，第二个FTM模块工作在正交解码器模式下，从两个90°移位的编码器信号A相和B相确定位置。如果使用霍尔传感器代替编码器，第二个FTM模块配置为输入捕获模式，第一个FTM模块的PWM信号根据霍尔传感器逻辑进行控制。有关更多详细信息，请参阅带有霍尔效应传感器的BLDC电机控制— *Kinetis MCUs* （文档 AN4376）。

3. FTM 操作模式

3.1. 边沿对齐PWM（EPWM）模式

在边沿对齐PWM模式下，FTM计数器从FTM_CNTIN值向上计数到FTM_MOD值。当FTM计数器从MOD值变为CNTIN值时，所有FTM通道的信号都在边缘对齐。

当QUADEN = 0, DECAPEN = 0, MCOMBINE = 0, COMBINE = 0, CPWMS = 0和MSnB = 1时，选择边沿对齐PWM模式。

边沿对齐PWM周期由MOD – CNTIN + 0x0001决定，脉冲宽度（或占空比）由 CnV – CNTIN or MOD – CNTIN – CnV决定，具体取决于控制位 ELSnB:ELSnA。

示例1 . 边沿对齐PWM

```
void Edge_Align_PWM_Init()
{
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
    /* Select and enable clock for FTM0 */
    PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

    /* Set PORTD pins for FTM0 */
    PORTD->PCR[15] = PORT_PCR_MUX(2); // FTM0, Channel0 PORTD-
    >PCR[16] = PORT_PCR_MUX(2); // FTM0, Channel1 PORTD->PCR[0] =
    PORT_PCR_MUX(2); // FTM0, Channel2 PORTD->PCR[1] =
    PORT_PCR_MUX(2); // FTM0, Channel3

    /* Enable registers updating from write buffers */ FTM0->MODE =
    FTM_MODE_FTMEN_MASK;
   /* Enable sync, combine mode and dead-time for pair channel n=1 and n=2 */ FTM0->COMBINE =
   FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK
                          | FTM_COMBINE_COMP1_MASK | FTM_COMBINE_DTEN1_MASK;
    /* Set Modulo in initialization stage (10kHz PWM frequency @112MHz system clock) */ FTM0->MOD =
    FTM_MOD_MOD(11200-1);
    /* 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[2].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
```

```
    FTM0->CONTROLS[3].CnSC = FTM_CnSC_MSB_MASK | FTM_CnSC_ELSB_MASK;
    /* Set channel value in initialization stage */
    FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(5600); // 50% duty cycle FTM0-
    >CONTROLS[1].CnV=FTM_CnV_VAL(5600);  // 50% duty cycle FTM0-
    >CONTROLS[2].CnV=FTM_CnV_VAL(2800);  // 25% duty cycle FTM0-
    >CONTROLS[3].CnV=FTM_CnV_VAL(2800); // 25% duty cycle
    /* Reset FTM counter */ FTM0-
    >CNT = 0;
    /* Insert deadtime (1us) */
    FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(7);
    /* 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;
}
```

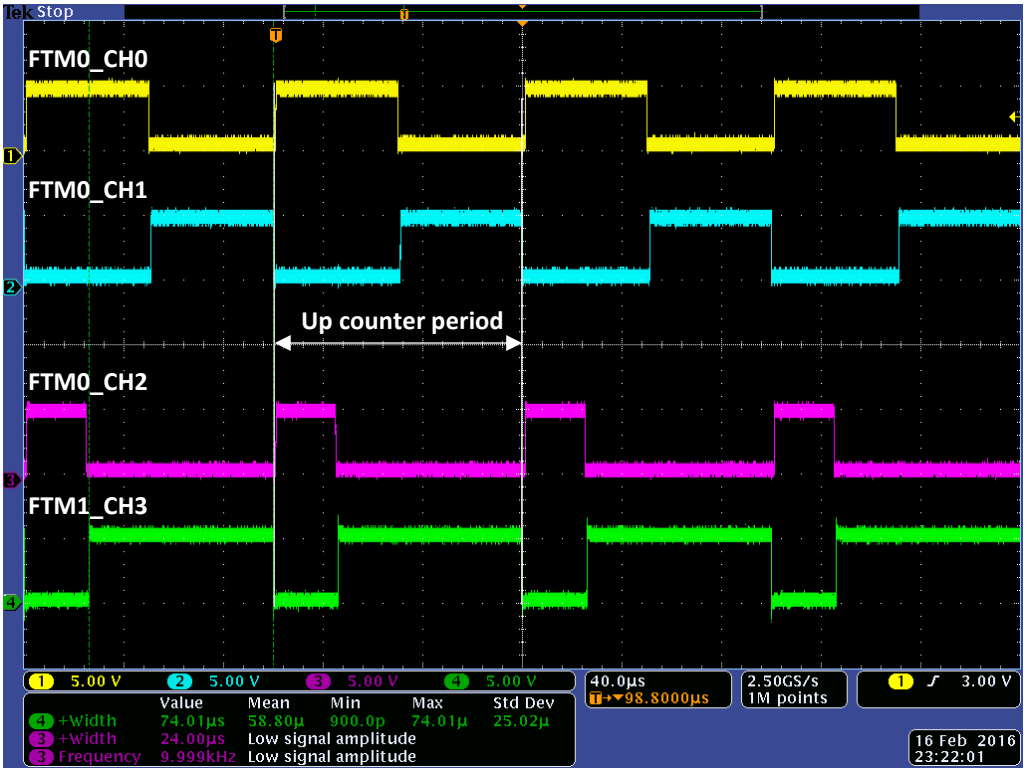

图1 . 边沿对齐PWM

在 图 1 中，示波器通道CH1, CH2, CH3和 CH4 分别代表FTM模块的FTM_CH0, FTM_CH1, FTM_CH2和 FTM_CH3。第一个通道对（FTM_CH0/FTM_CH1）和第二个通道对（FTM_CH2/FTM_CH3）在互补模式下工作，分别具有50%和25%的占空比。第二个通道对（FTM_CH2/FTM_CH3）演示了边缘对齐。

3.2. 中心对齐PWM（CPWM）模式

在中心对齐PWM模式下，FTM计数器从FTM_CNTIN向上计数到FTM_MOD，然后从FTM_MOD向下计数到FTM_CTNIN。所有FTM通道的信号在FTM计数器达到FTM_MOD值的点对齐。当 QUADEN = 0, DECAPEN = 0, MCOMBINE = 0, COMBINE = 0和CPWMS = 1时，选择中心对齐PWM模式。

中心对齐PWM的占空比确定为 2 × (CnV − CNTIN)。周期确定为 2 × (MOD − CNTIN)。

示例2 . 中心对齐PWM

```
void Center_Align_PWM_Init()
 {
      /* Enable clock for PORTD */
      PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
      /* Select and enable clock for FTM0 */
      PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

      /* Set PORTD pins for FTM0 */ PORTD-
      >PCR[15] = PORT_PCR_MUX(2); PORTD-
      >PCR[16] = PORT_PCR_MUX(2); PORTD-
      >PCR[0] = PORT_PCR_MUX(2); PORTD-
      >PCR[1] = PORT_PCR_MUX(2);
      /* Select up-down counter for Center-Align PWM */ FTM0->SC =
      FTM_SC_CPWMS_MASK;
      /* Combine mode and dead-time enable for channel0 and channel1 */ FTM0-
      >COMBINE = FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK;
      /* Combine mode and dead-time enable for channel2 and channel3 */ FTM0-
      >COMBINE |= FTM_COMBINE_COMP1_MASK | FTM_COMBINE_DTEN1_MASK;
      /* Set Modulo (10kHz PWM frequency @112MHz system clock) */ FTM0->MOD
      = FTM_MOD_MOD(5600-1);
      /* 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(2800); // 50% duty cycle FTM0-
      >CONTROLS[1].CnV=FTM_CnV_VAL(2800);  // 50% duty cycle FTM0-
      >CONTROLS[2].CnV=FTM_CnV_VAL(1400);  // 25% duty cycle FTM0-
      >CONTROLS[3].CnV=FTM_CnV_VAL(1400); // 25% duty cycle
      /* FTM counter reset */ FTM0-
      >CNT = 0;
      /* Insert DeadTime (1us) */
      FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(7);
      /* 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;

 }
```

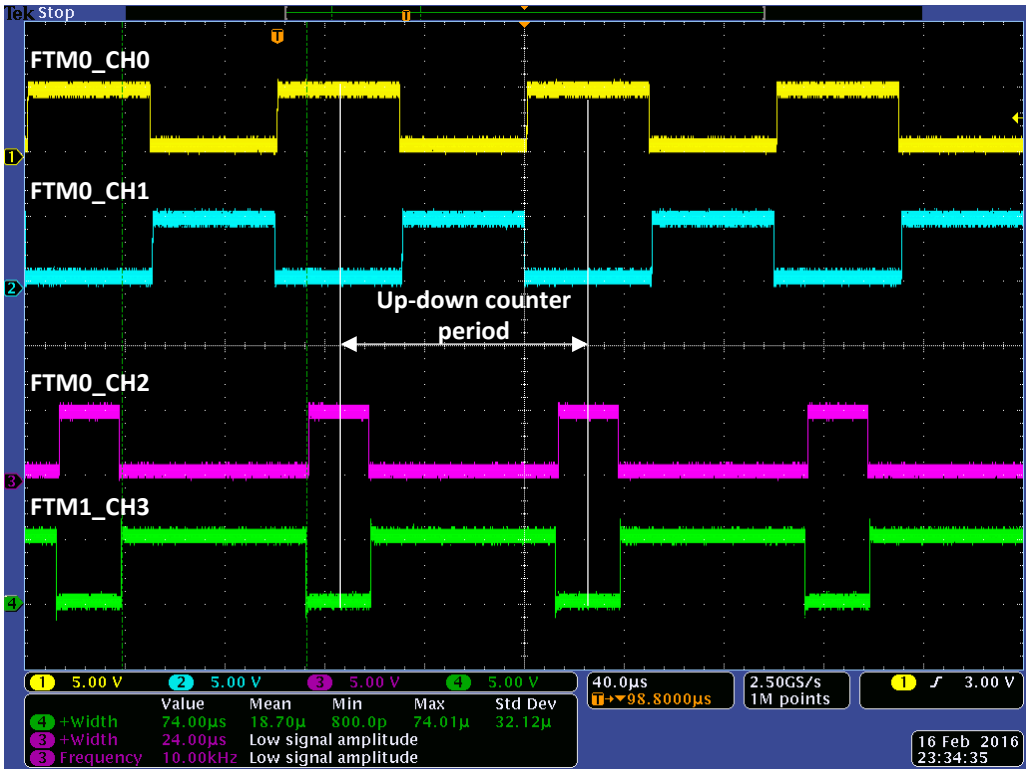

图2 . 中心对齐PWM

在 图 2中，示波器通道CH1、CH2、CH3和CH4分别代表FTM模块的FTM_CH0、FTM_CH1、FTM_CH2和FTM_CH3。第一个通道对（FTM_CH0/FTM_CH1）和第二个通道对（FTM_CH2/FTM_CH3）在互补模式下工作，分别具有50%和25%的占空比。第二个通道对（FTM_CH2/FTM_CH3）演示了中心对齐。

注意

向上/向下计数模式专用于生成中心对齐PWM，也可以使用向上计数模式，但是FTM通道必须配置为在组合模式下工作，有关更多信息，请参见 Section 3.4, “Combine mode”。

3.3. 互补模式和死区时间插入

FTM模式支持互补模式。如果通过FTM_COMBINE寄存器中的COMP位启用互补模式，则输出信号仅由偶数FTM通道生成。奇数输出信号由互补逻辑生成，作为偶数FTM通道的补码。对于FTM的每个通道对，是否输出互补信号是可以单独配置的。

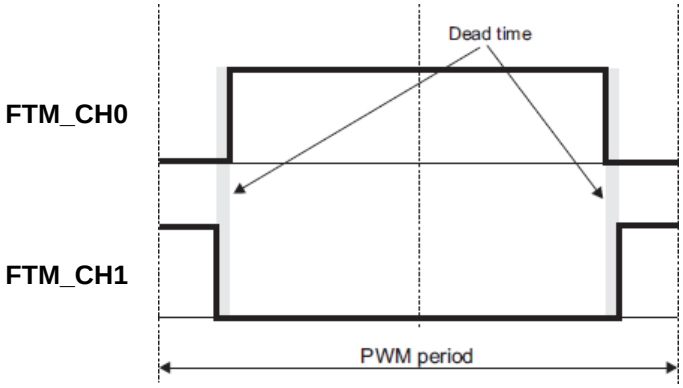

图3 . 带死区时间插入的通道对FTM_CH0/FTM_CH1的互补模式

为避免短路，死区时间必须插入互补信号中。死区插入由死区逻辑提供，遵循互补逻辑。该功能可以通过FTMxCOMBINEm寄存器中的DTEN位使能。死区逻辑将每个上升沿延迟一段时间，该时间由FTM_DEADTIME寄存器设置。死区时间由两部分组成，前两个最高有效位DTPS[1:0]定义系统时钟的预分频器。位DTVAL[5:0]使用预分频时钟定义占空比值。

互补模式和死区插入应用于边沿对齐PWM和中心对齐PWM模式，如前几节所述。

3.4. 组合模式

组合模式提供了更高的灵活性，因为PWM通道（n）输出是通过组合偶数通道（n）和相邻的奇数通道（n+1）产生的。这意味着偶数和奇数通道必须在互补模式下工作。

组合模式仅允许使用递增计数器、非对称PWM或相移PWM来生成EPWM和CPWM。相移PWM通常用于移相全桥转换器和电机控制应用，其中三相定子电流由采样电阻得到。有关更多详细信息，请参阅使用Kinetis KEA128的三相 BLDC电机无传感器控制参考设计（文档DRM151）。

当QUADEN = 0, DECAPEN = 0, MCOMBINE = 0, COMBINE = 1和CPWMS = 0时，选择组合模式。

要生成具有高真脉冲的相移PWM，请将控制位设置为ELSnB:ELSnA = 1：0。此代码示例显示了用于生成相移PWM的FTM0模块的配置。

示例3 . 移相PWM

```
void Phase_Shifted_PWM()
{
    /* Enable clock for FTM1 */
    PCC->PCCn[PCC_FLEXTMR1_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;
    /* Enable clock for PORTB */
    PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
```

```
    PORTB->PCR[2] = PORT_PCR_MUX(2); // Set PTB2 for FTM1 – Channel0 PORTB-
    >PCR[3] = PORT_PCR_MUX(2); // Set PTB3 for FTM1 – Channel1 PORTD->PCR[8] =
    PORT_PCR_MUX(6); // Set PTD8 for FTM1 – Channel4 PORTD->PCR[9] =
    PORT_PCR_MUX(6); // Set PTd9 for FTM1 –Channel5

      /* Enable combine, complementary mode and dead-time for channel pair CH0/CH1 and CH4/CH5 */ FTM1->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;

    FTM1->CONTROLS[0].CnSC=FTM_CnSC_ELSB_MASK;  // Select high-true pulses FTM1-
    >CONTROLS[1].CnSC=FTM_CnSC_ELSB_MASK;  // Select high-true pulses FTM1-
    >CONTROLS[4].CnSC=FTM_CnSC_ELSB_MASK;  // Select high-true pulses FTM1-
    >CONTROLS[5].CnSC=FTM_CnSC_ELSB_MASK; // Select high-truepulses
    /* Set Modulo (10kHz PWM frequency @112MHz system clock) */ FTM1->MOD =
    FTM_MOD_MOD(11200-1);                                   // Set modulo
    FTM1->CONTROLS[0].CnV=FTM_CnV_VAL(2800);  // Set channel Value FTM1-
    >CONTROLS[1].CnV=FTM_CnV_VAL(8400);  // Set channel Value FTM1-
    >CONTROLS[4].CnV=FTM_CnV_VAL(5600);  // Set channel Value FTM1-
    >CONTROLS[5].CnV=FTM_CnV_VAL(11200); // Set channel Value FTM1->CNT = 0;
                                                            // Counter reset
    /* Insert DeadTime (1us) */
    FTM1->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(7);
    FTM1->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
}
```

代码示例的结果行为如下图所示：

图 4. Phase-shifted PWM

在 图 4中，示波器通道 CH1、CH2、CH3和CH4分别代表FTM1模块通道FTM1_CH0、FTM1_CH1、FTM_CH4和FTM_CH5。第一个通道对（FTM_CH0/FTM_CH1）和第二个通道对（FTM_CH4/FTM_CH5）都在互补模式下工作，占空比为50%。第二个通道对（FTM_CH4/FTM_CH5）与第一个通道对（FTM_CH0/FTM_CH1）相移90°。

图 4 还显示了PWM生成的灵活性，因为50%的占空比可以根据特定的应用要求进行 边沿对齐、中心对齐或各种偏移。

3.5. 单边捕获模式

FTM 捕获模式主要用于测量信号的脉冲宽度或周期。FTM 捕获模式的另一个选择是检测外部信号的上升/下降沿并产生中断以通知外部事件出现。FTM 捕获模式也用于BLDC电机控制应用，其中霍尔传感器用于检测转子位置并计算转子速度，从而可以稳定速度环。霍尔传感器连接到独立的FTM (FTM_CHx) 的通道。然后，FTM 可以检测霍尔传感器信号的下降沿和上升沿并生成捕获中断。在捕获中断程序中，PWM 信号的占空比会根据霍尔传感器逻辑进行更新。

当 DECAPEN = 0、MCOMBINE = 0、COMBINE = 0、CPWMS = 0、MSnB:MSnA = 0:0 且 ELSnB:ELSnA ≠ 0:0 时，选择单边沿捕获模式。

要测量信号的脉冲宽度或周期，请选择FTM模块 FTM_CHx 的输入通道，并通过控制位 ELSnB:ELSnA 选择边沿触发。当通道输入上出现所选边沿时，FTM计数器的当前值将被存储到 CnV 寄存器中并产生通道中断（如果 CH(n)IE = 1）。在例程中断中，将 CnV 寄存器的值保存到一个变量中，并将当前值与前一个中断例程中保存的值进行区分。如果所选捕获模式在上升沿 (ELSnB:ELSnA= 0:1) 或下降沿 (ELSnB:ELSnA= 1:0) 触发，则差值等于信号周期。如果所选的捕获模式在两个边沿都触发 (ELSnB:ELSnA = 1:1)，则差值等于被测信号的脉冲宽度。

此示例显示FTM0的输入捕获模式，该模式测量通过FTM0_CH0通道的输入信号的时间周期：

示例4 . 单边沿捕获模式

```
void FTM0_Single_Edge_Capture_Mode()
{
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
    /* Select and enable clock for FTM0 */
    PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

    PORTD->PCR[15] = PORT_PCR_MUX(2);                          // Set PTD15 for FTM0 - Channel0
    FSL_NVIC->ISER[FTM0_IRQn / 32] |= (1 << (FTM0_IRQn % 32)); // Enable FTM0 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);
}

/* Interrupt routine to determine period of tested signal */
void FTM0_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
```

}

图5．FTM0_CH0输入信号

在图 5中，示波器通道CH0表示输入到FTM0_CH0通道的测试信号。每次出现上升沿，都会产生捕获中断，并在中断程序中计算信号周期，如 Example 4所示。结果显示在S32设计工作室中，如下图：

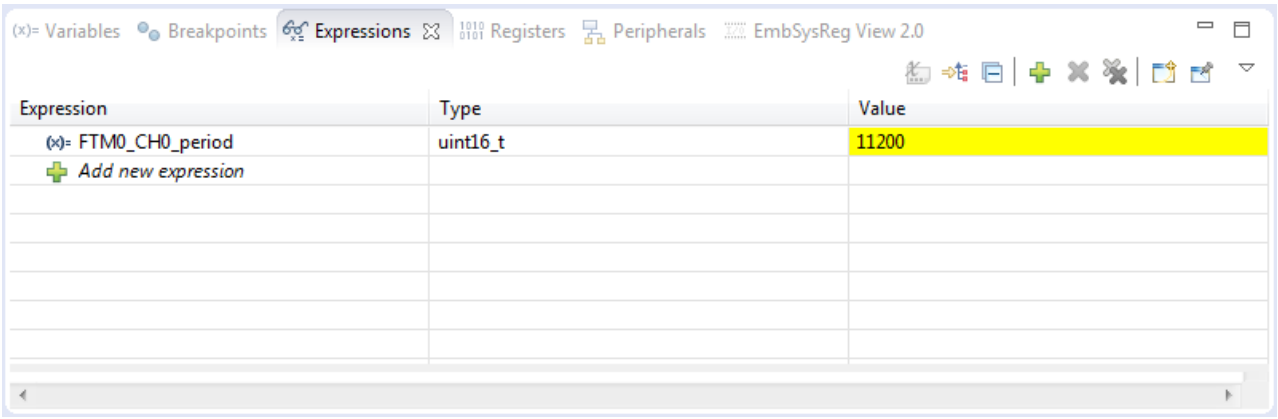

图6．以周期数显示的测试信号周期

测量信号的周期以周期数显示 (FTM0_CH0_period = 11200)。由于 FTM0 使用设置为 112 MHz 的系统时钟频率，因此可以将被测信号的周期转换为时间量（FTM0_CH0_period = 100 µs）。

3.6. 双边捕获模式

双边沿捕获模式使用两个 FTM 通道，可以测量信号的正极或负极脉冲宽度。在此模式下，信号必须通过偶数 FTM 通道输入，而忽略奇数通道。

DECAPEN = 1 时选择双边沿捕获模式。FTM 的双边沿捕获模式可以工作在一次性捕获模式或连续捕获模式。MS(n)A = 0 时选择一次性捕获模式。如果使能 DECAP 位，则捕获边沿。对于每次新的测量，必须清除 CH(n)F 和 CH(n+1)F，并且必须再次设置 DECAP 位。MS(n)A = 1 时选择连续捕获模式。在这种模式下，如果 DECAP 位被设置，则边沿被连续捕获。对于每次新的测量，都需要清除 CH(n)F 和 CH(n+1)F 位。

要测量被测信号的正极脉宽（无论是单次模式还是连续模式），通道（n）必须配置为捕获上升沿（ELS（n）B：ELS（n） A = 1:0) 和通道 (n+1) 必须配置为捕获下降沿 (ELS(n+1)B:ELS(n+1)A = 0:1)。当检测到被测信号的第二个下降沿时，CH(n+1)F 置位，DECAP 位清零，并产生中断（如果 CH(n+1)IE=1）。在中断程序中，将C(n+1)V和C(n)V寄存器中保存的值相减，确定被测信号的正极脉宽，并清除CH(n+1)F 位。

如果应用需要测量信号的负极性脉宽，则通道 (n) 必须配置为捕获下降沿 (ELS(n)B:ELS(n)A = 0:1)和通道(n) +1) 必须配置为捕获上升沿 (ELS(n+1)B:ELS(n+1)A = 1:0)。要确定被测信号周期，通道 (n) 和通道 (n+1) 必须在相同的边沿上敏感。

此示例显示了 FTM0 模块的双边沿捕获模式，该模式用于确定两个输入信号的正脉冲度。

示例5 . 双边沿捕获模式

```
void FTM0_Dual_Edge_Capture_Mode()
{
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
    /* Select and enable clock for FTM0 */
    PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

    PORTD->PCR[15] = PORT_PCR_MUX(2);          // Set PTD15 for FTM0 - Channel0
    PORTD->PCR[0] = PORT_PCR_MUX(2);           // Set PTD0 for FTM0 - Channel2

    FSL_NVIC->ISER[FTM0_IRQn / 32] |= (1 << (FTM0_IRQn % 32));          // Enable FTM0 interrupt

    /* 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);
}
```

```
/* Interrupt routine to measure positive polarity pulse width of tested input signals */
void FTM0_IRQHandler()
{
    if(FTM0->CONTROLS[1].CnSC & FTM_CnSC_CHF_MASK)          // wait for falling edge of FTM0_CH0 signal
    {
        CH0_edge1 = FTM0->CONTROLS[0].CnV;
        CH0_edge2 = FTM0->CONTROLS[1].CnV;
        FTM0_CH0_pulse_width = CH0_edge2 - CH0_edge1; // pulse width calculation of FTM0_CH0 signal FTM0-
        >CONTROLS[1].CnSC &= ~FTM_CnSC_CHF_MASK;            // clear FTM0_CH1 capture flag
    }
    if(FTM0->CONTROLS[3].CnSC & FTM_CnSC_CHF_MASK)          // wait for falling edge of FTM0_CH2 signal
    {
        CH2_edge1 = FTM0->CONTROLS[2].CnV;
        CH2_edge2 = FTM0->CONTROLS[3].CnV;
        FTM0_CH2_pulse_width = CH2_edge2 - CH2_edge1; // pulse width calculation of FTM0_CH2 signal

        FTM0->CONTROLS[3].CnSC &= ~FTM_CnSC_CHF_MASK;   // clear FTM0_CH3 capture flag
    }
}
```

图7 . FTM0_CH0和FTM0_CH2输入信号

图 7显示了通过 FTM0_CH0 和 FTM0_CH2 通道输入 FTM0 模块的两个测量信号。它们的正极性脉冲宽度为 17.8 µs 和 50 µs。示例5中代码的结果显示在S32设计工作室中：

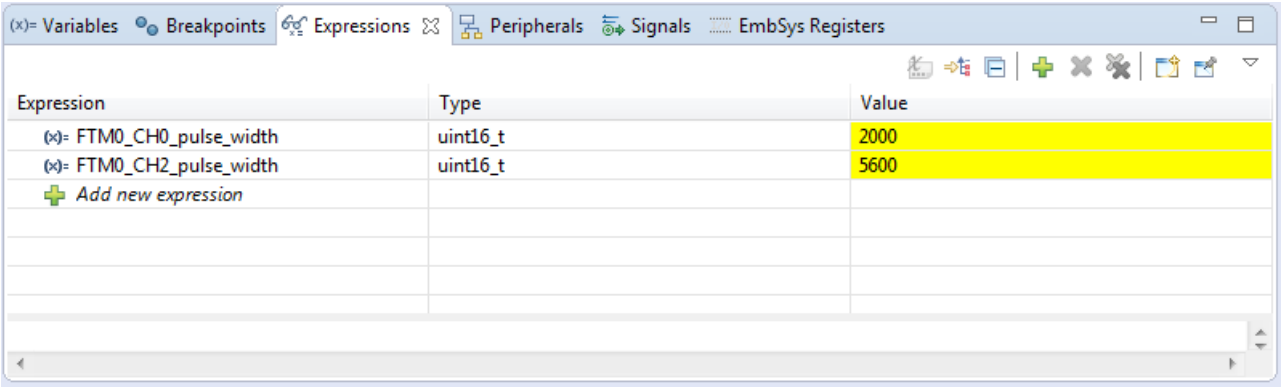

图8 . 以周期数显示的测试信号的正极性脉冲宽度

图8显示了以周期数显示的测试信号的脉冲宽度（FTM0_CH0_pulse_width=2000，FTM0_CH2_pulse_width=5600）。以112MHz时钟作为FTM0模块的时钟源，可以将被测信号的正极性脉冲宽度转换为时间量（FTM0_CH0_pulse_width=17.8μs，FTM0_CH2_pulse_width = 50 μs）。

3.7. 正交解码模式

FTM模块的正交解码器模式用于解码来自增量编码器传感器的信号，增量编码器传感器通常用于电机控制应用中的位置测量。增量式编码器通常基于光学技术。LED发出的光穿过金属盘中的插槽，并被光电晶体管检测到。1024脉冲编码器的输出信号如下图所示：

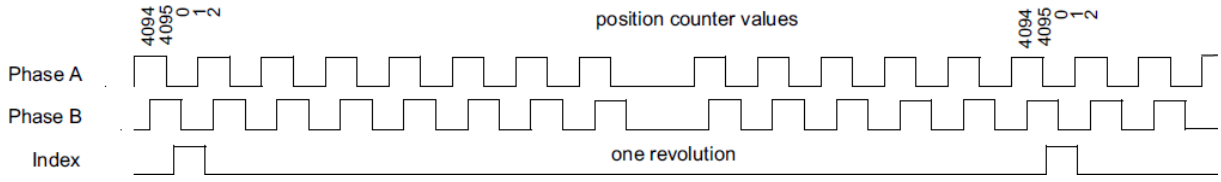

图9 . 编码器输出信号

有三个输出信号。A 相和 B 相信号由一系列相移 90 度的脉冲组成（因此使用术语“正交”）。第三个信号（称为“索引”）提供绝对位置信息。在运动控制中，用于检查脉冲计数的一致性。每次旋转后，都会捕获计数脉冲的值并将其与定义的值进行比较。如果检测到差异，则控制算法必须执行位置偏移补偿。

当前的S32K 设备配备 FTM 模块：FTM0、FTM1、FTM2 和 FTM3。虽然所有 FTM 都可以配置为生成 PWM 信号来控制电动机，但只有 FTM1 和 FTM2 模块可以在正交解码器模式下工作以测量转子位置。它们具有用于 A 相和 B 相信号的专用引脚。

QUADEN = 1 时选择正交编码器模式。正交编码器模式下可以使用两种子模式：计数和方向编码模式以及A 相和B 相编码模式。当 QUADMODE = 1 时，计数和方向编码模式使能。在此模式下，B 相输入指示计数方向，A 相输入定义计数速率。

要处理来自编码器传感器的 A 相和 B 相信号，请使能 A 相和 B 相编码模式 (QUADMODE = 0)。在此模式下，A 相和 B 相信号指示计数方向和计数速率。如果 B 相信号滞后于 A 相信号，则 FTM 计数器会在检测到两个信号的每个上升/下降沿后递增。如果 B 相信号领先于 A 相信号，则 FTM 计数器会在检测到两个信号的每个上升/下降沿后递减，并且 FTM_QDCTRL 寄存器中的 QUADIR 位指示计数方向。

此代码示例演示了A 相和 B 相编码子模式的正交解码器模式的功能：

示例6 . 正交解码器模式

```
void FTM2_Quadrature_Decoder_Mode()
{
      /* Enable clock for PORTD */
      PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
      /* Select and enable clock for FTM0 */
      PCC->PCCn[PCC_FLEXTMR2_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

      PORTD->PCR[10] = PORT_PCR_MUX(3); // Set PTD10 for FTM2 - Phase B input PORTD->PCR[11] =
      PORT_PCR_MUX(3); // Set PTD11 for FTM2 - Phase A input PORTD->PCR[0] = PORT_PCR_MUX(1);
      PTD->PDDR=1<<0;

      FSL_NVIC->ISER[FTM2_IRQn / 32] |= (1 << (FTM2_IRQn % 32));                    // Enable interrupt

      /* Encoder simulation with totally 10 rising/falling edges */ FTM2->MOD =
      FTM_MOD_MOD(10);
      FTM2->CNTIN = FTM_CNTIN_INIT(0);
      FTM2->QDCTRL= FTM_QDCTRL_QUADEN_MASK;

      FTM2->CNT = 0;
      /* Select clock */
      FTM2->SC = FTM_SC_CLKS(1) | FTM_SC_TOIE_MASK;
}

/* Timer overflow interrupt routine indicating one mechanical revolution */
void FTM2_IRQHandler()
{
      PTD->PTOR|=1<<0;
      FTM2->SC &= ~FTM_SC_TOF_MASK;
}
```

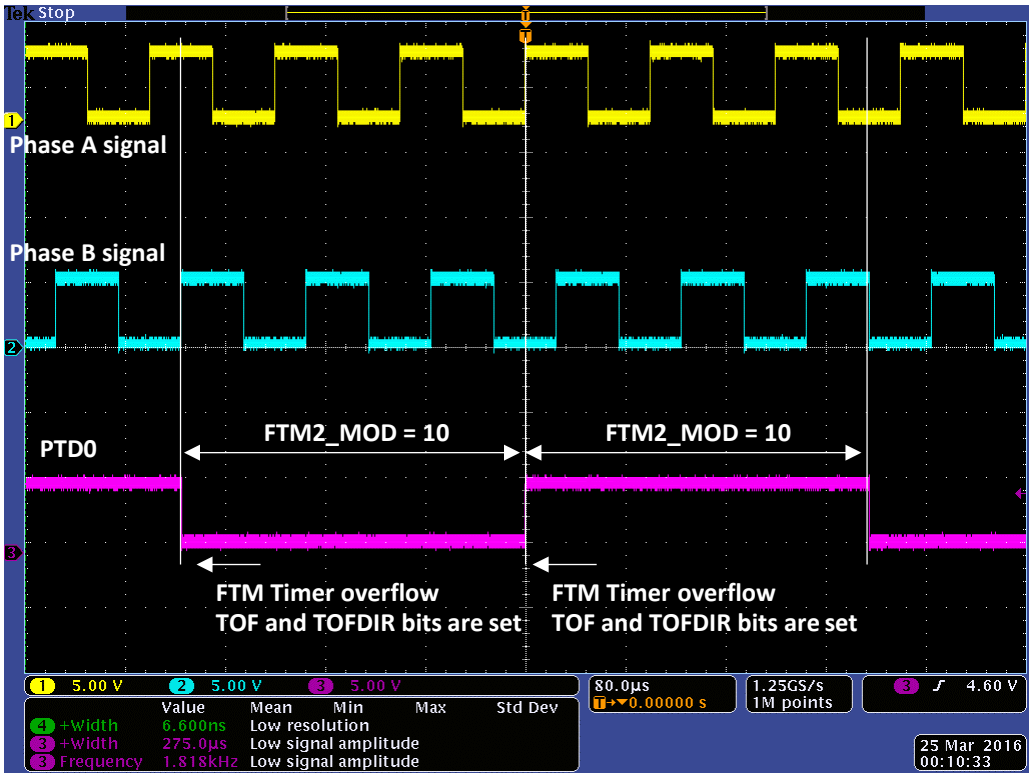

图10 . 正交解码器模式

图 10 中，示波器通道 CH1 和 CH2 显示输入到 FTM2 模块的编码器信号 A 相和 B 相。示波器通道CH3代表每次FTM计数器达到MODULO寄存器的值时产生的FTM2计数器溢出中断（每次检测到A相和B相信号的10个上升/下降沿）。当 FTM2 定时器溢出时，TOF 位和 TOFDIR 被置位，因为 B 相信号滞后于 A 相信号（如图 10所示）。

4. FTM 特征

4.1. 屏蔽、反相和软件控制功能

屏蔽和软件控制功能强制 FTM 通道的输出处于非活动/需要状态，保持通道输出的恒定逻辑。 这些 FTM 功能可以通过软件打开/关闭电源设备。反相功能翻转 FTM 通道对的信号。

屏蔽功能用于 BLDC 电机控制应用。三相 BLDC 电机通常由配备六个 MOSFET 或六个 IGB（绝缘栅双极晶体管）的三相功率逆变器供电。BLDC 电机常用的控制技术是 6 步换向法。在这种控制技术中，两相通电，第三相断开。因此，必须关闭相应逆变器支路的两个功率器件。FTM 模块的屏蔽特性可以在这个应用程序中实现来实现这样的功能。 每个 FTM 通道在FTM_OUTMASK 寄存器中有一个对应的位。对 OUTMASK 寄存器的任何写操作只是将值锁存到其写缓冲区中。

当FTM时钟被禁用（在FTM_SYNC[SYNCHOM]=0时在FTM输入时钟的每个上升沿）或着在软件或硬件同步更新FTM_SYNCONF 寄存器中的 SWOM 位或 HWOM 位时候，FTM 通道的输出逻辑可以在写入 FTM_OUTMASK 寄存器后立即更新（参见Section 4.3, “Updating FTM registers”）。当 FTM_CHx 通道被屏蔽时，根据 FTM_POL 寄存器中使能的位来控制输出逻辑保持高/低。

软件控制功能通过设置或清除 FTM_SWOCTRL 寄存器中的 CHxOC 位来强制 FTM 通道的输出为软件定义的值。如果 CHxOC 位清零，FTM 通道的输出仍由 PWM 或其他源驱动。如果CHxOC 位被置位，相应的 FTM 通道的输出受软件影响。FTM_SWOCTRL 寄存器中的 CHxOCV 位控制各个 FTM 通道的逻辑。当 CHxOCV = 1 时，FTM_CHx 通道的输出为1，否则为 0。

反相功能通常用于直流电机控制应用。直流电机由 H 桥供电，一个通道对(FTM_CH0/CH1) 连接到一个分支，而另一对通道 (FTM_CH2/CH3) 连接到另一个分支。两个通道对都在互补模式下工作。此类应用要求对角线功率器件同时开启/关闭，这意味着 FTM_CH0 和 FTM_CH3 必须具有相同的时序。为了保证这样的条件，FTM_CH1 和 FTM_CH3 必须工作在反相模式。 FTM_INVCTRL 寄存器中的 INVmEN 位使能相应通道对的反相。从写入缓冲区的值更新FTM_INVCTRL 寄存器的值与更新FTM_OUTMASK 寄存器值的方式相同。

Section 4.3.3, “Updating FTM registers by software trigger” 中演示了屏蔽功能。

4.2. 故障控制功能

FTM 模块的故障控制在电机控制应用中发挥着重要作用，因为它提供了在关键时刻保护功率设备以及整个电力驱动系统的机会，当出现过温、过压、 或发生过电流。在这种情况下，故障信号可以通过传感器或特殊电路产生。一旦在 FTM 故障引脚的输入端检测到故障信号，故障控制就能够停止所有 PWM 通道。接收到故障信号后可以产生中断以减轻故障损失。

所有 FTM 中断源（故障中断、FTM 计数器溢出和重装载中断、通道比较事件中断和捕获中断）共享相同的中断向量号。当出现故障信号时，FTM 通道的输出被禁用并保持在 FTM_POL 寄存器中定义的安全逻辑电平。例如，如果 POL0 = 1并且存在故障，则 FTM_CH0 被禁用并强制为高电平。相反，如果 POL0 = 0 并且存在故障，则 FTM_CH0 被禁用，但被强制为低电平。

FTM 有多个通道。然而，FTM 不能通过特定的故障信号来禁用特定的通道。故障信号是由所有进入的故障信号OR 运算的结果。FTM 通道是否可以被故障信号禁用取决于 FTM_COMBINE 寄存器中的 FAULTENx 位。由此产生的故障信号可以禁用所有 FTM 通道或仅禁用偶数通道 (FTM_CH0/CH2/CH4/CH6)，这取决于 FTM_MODE 寄存器中的 FAULTM 位字段。

PWM通道的输出恢复有两种方式：FAULTM[1:0]=11设置的自动故障清除和FAULTM[1:0]=0:1或1:0时的手动故障清除。从下一个PWM周期中的安全状态恢复PWM信号，在选择自动故障清除模式时，无需任何软件干预。要在手动清除模式下恢复 PWM 信号，必须清除 FTM_MODE 寄存器中的 FAULTF 位，然后在下一个 PWM 周期使能 PWM。

在以下代码示例中，启动了两个 FTM 模块以演示故障控制功能。FTM0 配置为中心对齐 PWM模式，FTM1 用于生成故障信号，该信号通过故障引脚 FTM0_FLT0 从外部影响 FTM0 通道的输出。

示例7．中心对齐PWM的故障控制

```
/* FTM0 configured to generate Center-Align PWM with fault control */
void CPWM_and_Fault_Control()
{
    /* Enable clock for PORTA */
    PCC->PCCn[PCC_PORTA_INDEX] = PCC_PCCn_CGC_MASK;
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
    /* Select and enable clock for FTM0 */
    PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

    PORTD->PCR[15] = PORT_PCR_MUX(2); // Set PTD15 for FTM0 - Channel0 PORTD->PCR[16] =
    PORT_PCR_MUX(2); // Set PTD16 for FTM0 - Channel1 PORTD->PCR[0] = PORT_PCR_MUX(2);
    // Set PTD0 for FTM0 - Channel2 PORTD->PCR[1] = PORT_PCR_MUX(2); // Set PTD0 for FTM0 -
    Channel3
    PORTA->PCR[14] = PORT_PCR_MUX(2); // Set PTD14 as a fault pin

    /* Enable registers updating from write buffers */ FTM0->MODE =
    FTM_MODE_FTMEN_MASK;
    /* Enable combine, complementary mode and dead-time for pair channel n=1 and n=2 */ FTM0-
    >COMBINE=FTM_COMBINE_COMBINE0_MASK|FTM_COMBINE_COMP0_MASK|FTM_COMBINE_DTEN0_MASK
                    |FTM_COMBINE_FAULTEN0_MASK|FTM_COMBINE_COMBINE1_MASK|FTM_COMBINE_COMP1_MASK
                    |FTM_COMBINE_DTEN1_MASK|FTM_COMBINE_FAULTEN1_MASK;
    /* Set Modulo (10kHz PWM frequency @112MHz system clock) */ FTM0->MOD
    = FTM_MOD_MOD(11200-1);
    /* 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(1120);
    FTM0->CONTROLS[1].CnV=FTM_CnV_VAL(11200-1120);
    FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(2800);
    FTM0->CONTROLS[3].CnV=FTM_CnV_VAL(11200-2800);
    /* FTM counter reset */ FTM0-
    >CNT = 0;
    /* Insert DeadTime (1us) */
    FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(7);
    /* Set PTA14 pin as a fault input FTM0_FLT0 */ FTM0-
    >FLTCTRL = FTM_FLTCTRL_FAULT0EN_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);
    /* A 1 at the fault input indicates the fault */ FTM0->FLTPOL &=
    ~FTM_FLTPOL_FLT0POL_MASK;
    /* Select clock and enable PWM */
    FTM0->SC = FTM_SC_CLKS(1) | FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN2_MASK
                |FTM_SC_PWMEN3_MASK;
}

/* FTM1 is configured to generate fault signals for FTM0 module */
```

```
void Fault_Signals_Generator()
{
    /* Enable clock for FTM1 */
    PCC->PCCn[PCC_FLEXTMR1_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;
    /* Enable clock for PORTB */
    PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;

    PORTB->PCR[2] = PORT_PCR_MUX(2);                    // Set PTB2 for FTM1 – Channel0

    FTM1->SC = FTM_SC_CPWMS_MASK;                       // Select up-down counter for Center-Align PWM FTM1-
    >CONTROLS[0].CnSC = FTM_CnSC_ELSB_MASK;             // Select high-true pulses
    /* Set Modulo (2240Hz PWM frequency @112MHz system clock) */ FTM1->MOD =
    FTM_MOD_MOD(50000-1);                               // Set modulo
    FTM1->CONTROLS[0].CnV = FTM_CnV_VAL(25000);         // Set channel Value
    FTM1->CNT = 0;                                      // Counter reset
    FTM1->SC |= FTM_SC_CLKS(1)|FTM_SC_PWMEN0_MASK;      // Select clock and enable PWM
}
```

代码示例运行的结果如下图所示：

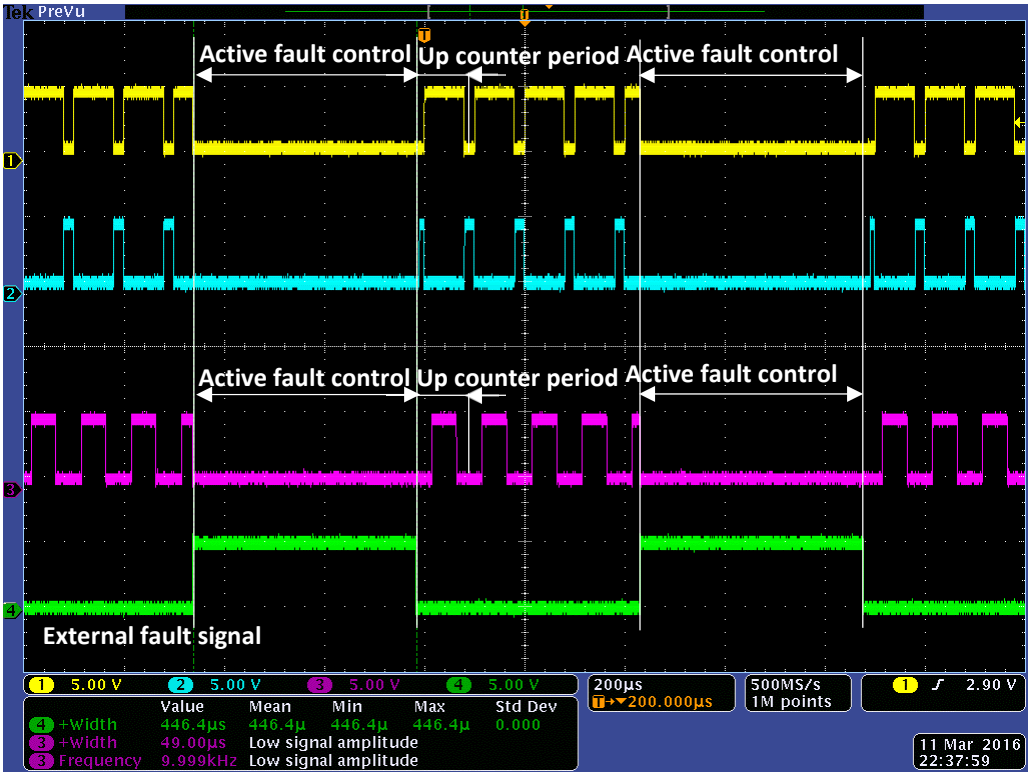

图11 . 中心对齐PWM的故障控制

在图 11中，示波器通道 CH1、CH2 和 CH3 分别代表 FTM0 模块通道 FTM0_CH0、FTM0_CH1 和 FTM0_CH2。FTM0_CH0 和 FTM0_CH1 工作在互补和组合模式，与 FTM0_CH2 和 FTM0_CH3 相同。通道 CH4 显示外部故障信号（由 FTM1 产生），其频率为 FTM0 频率的十分之一。当在 FTM0_FLT0 故障输入引脚上检测到上升沿时，所有 FTM0 通道都被强制为安全（低）逻辑。图 11 还显示，当故障输入信号为 0 时，所有 FTM0 通道都在下一个计数器周期恢复。

4.3. 更新FTM寄存器

在电机控制或开关电源应用中，施加在负载上的电压必须由 PWM 占空比控制。PWM 占空比可以通过在 FTM_CnV 和 FTM_CNTIN 寄存器中设置适当的值来控制。某些应用还需要更改 PWM 的周期。该周期由 FTM_MOD 寄存器中的值控制。

FTM_CnV、FTM_CNTIN 和 FTM_MOD 是双缓冲寄存器，这意味着对这些寄存器的写入只是将它们的值锁存到它们的写入缓冲区中。FTM 模块的双缓冲寄存器有通道 (n) 值 (FTM_CnV)、计数器初始值 (FTM_CNTIN)、模数 (FTM_MOD)、半周期寄存器 (FTM_HCR)、输出掩码 (OUTMASK)、软件输出控制 (SWOCTRL) 和反相控制寄存器 (INVCTRL)。然后可以根据选择的更新方案用它们的缓冲值更新双缓冲寄存器。S32K 器件上的当前实现更新寄存器方法有使用半周期或全周期重载策略、软件或硬件 PWM 同步，或者直接禁用缓冲器。下表总结了所有双缓冲寄存器及其更新方法：

表3 . 更新双缓冲FTM寄存器

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 CLKS[1:0] ≠ 0:0, FTMEN=1,	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. For the up/down-counting mode, see Table 4.

表4 . 向上/向下计数模式下的重新加载机会

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.

Table 3 总结了所有双缓冲寄存器及其更新方法。更新方法的配置步骤将在以下部分详细讨论。

4.3.1. 在初始化阶段更新FTM寄存器

在初始化阶段，当 FTM 模块被禁用（CLKS[1:0] = 0:0）时，FTM 寄存器在写入 FTM_CnV、FTM_CNTIN 和 FTM_MOD 寄存器后立即更新。在初始化步骤之后，FTM 模块通过设置 CLKS[1:0] ≠ 0:0 并使能 FTM_SC 寄存器中的 PWMENn 位来开始生成 PWM（参见 Example 1 或 Example 2）。

4.3.2. 半周期和全周期重装策略

要在 FTM 计数器运行时更改 PWM 占空比或 PWM 时间周期，可以应用半周期和全周期重新加载策略。此功能允许使用其缓冲区的内容更新 FTM 寄存器，具体取决于通过设置 FTM_PWMLOAD 寄存器中的 LDOK 位选择的重新加载机会。当发生重载机会时，FTM_SC 寄存器中的 RF 位被置位，如果 RIE = 1，则产生重载机会中断。在中断程序中，FTM 寄存器可以同时改变和更新。

如果选择递增计数模式生成边沿对齐 PWM，半周期和全周期重载机会可以根据以下步骤更新 FTM 寄存器：

1. 初始化FTM0产生边沿对齐PWM并使能FTM_MODE寄存器中的FTMEN位和FTM_SC寄存器中的RIE位。
2. 使能FTM_PWMLODE寄存器中的HCSEL位，将FTM_HCR寄存器中的值调整为MOD/2表示半周期重载，HCSEL = 0表示全周期重载。
3. 在中断中，更改FTM寄存器的值以更新其缓冲区中的值并清除FTM_SC寄存器中的RF位。

当选择向上/向下计数模式以生成中心对齐PWM时，必须考虑不同的情况：

1. 初始化FTM0，产生中心对齐PWM，使能FTM_MODE寄存器中的FTMEN位和FTM_SC寄存器中的RIE位。
2. E为半周期重载机会使能FTM_SYNC寄存器中的CNTMIN和CNTMAX位，为全周期重载使能CNTMIN = CNTMAX = 0。
3. 在中断中，更改FTM寄存器的值以更新其缓冲区中的值并清除FTM_SC寄存器中的RF位。

以下代码列举了中心对齐 PWM 模式和半周期重装载的 FTM0 部分初始化函数以及中断服务函数，其中通道的值 C0V 和 C2V 根据变量标志进行修改。

示例8 . 半/全周期重装策略

```
void Center_Align_PWM_Init()
{
    ...

    /* Enable half cycle reload */
    FTM0->SYNC = FTM_SYNC_CNTMAX_MASK | FTM_SYNC_CNTMIN_MASK;
    /* 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_PWMEN2_MASK |
                | FTM_SC_PWMEN3_MASK | FTM_SC_RIE_MASK;
}

/* Half cycle reload opportunity interrupt routine */
void FTM0_IRQHandler()
{
    if(flag)
    {
        FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(5600);
        FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(2800);
    }
    else
    {
        FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(2800);
        FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(5600);
    }
        flag = !flag;
        PTD->PTOR |= 1<<3;                          // Toggle PTD3 to show reload opportunities FTM0-
        >PWMLOAD |= FTM_PWMLOAD_LDOK_MASK;          // Set LDOK bit
        FTM0->SC &= ~FTM_SC_RF_MASK;                // Clear Timer Flag
}
```

下图展示了执行中心对齐PWM模式下的FTM寄存器的半周期和全周期重载策略。

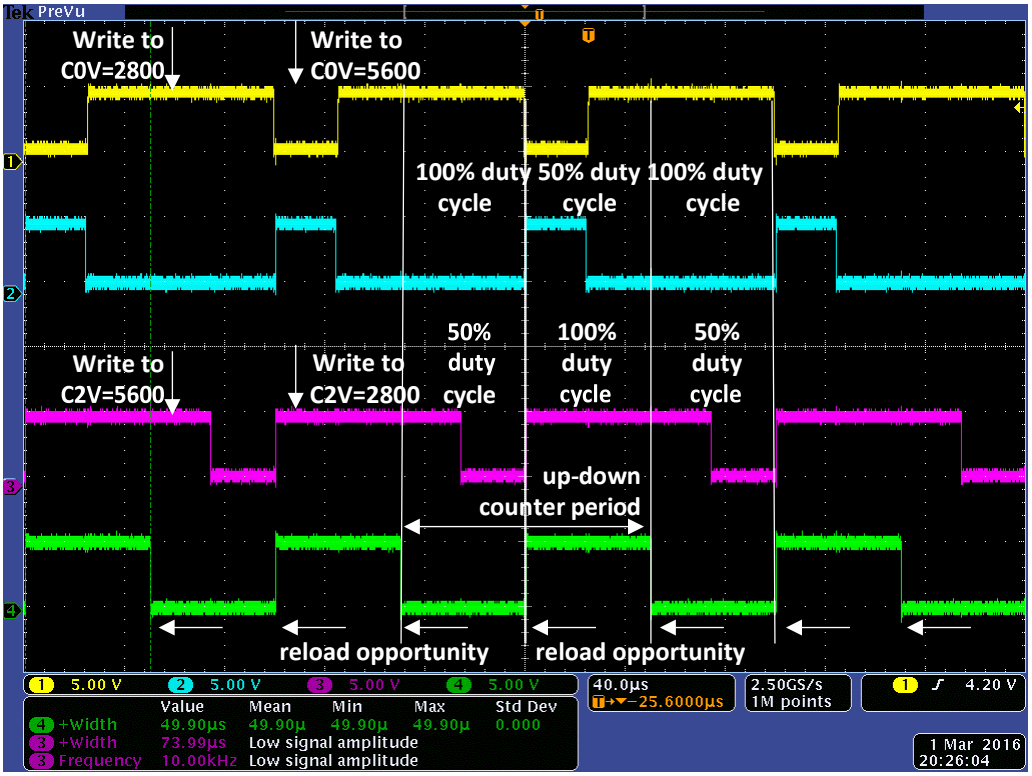

图12 . 中心对齐PWM和半周期重载策略

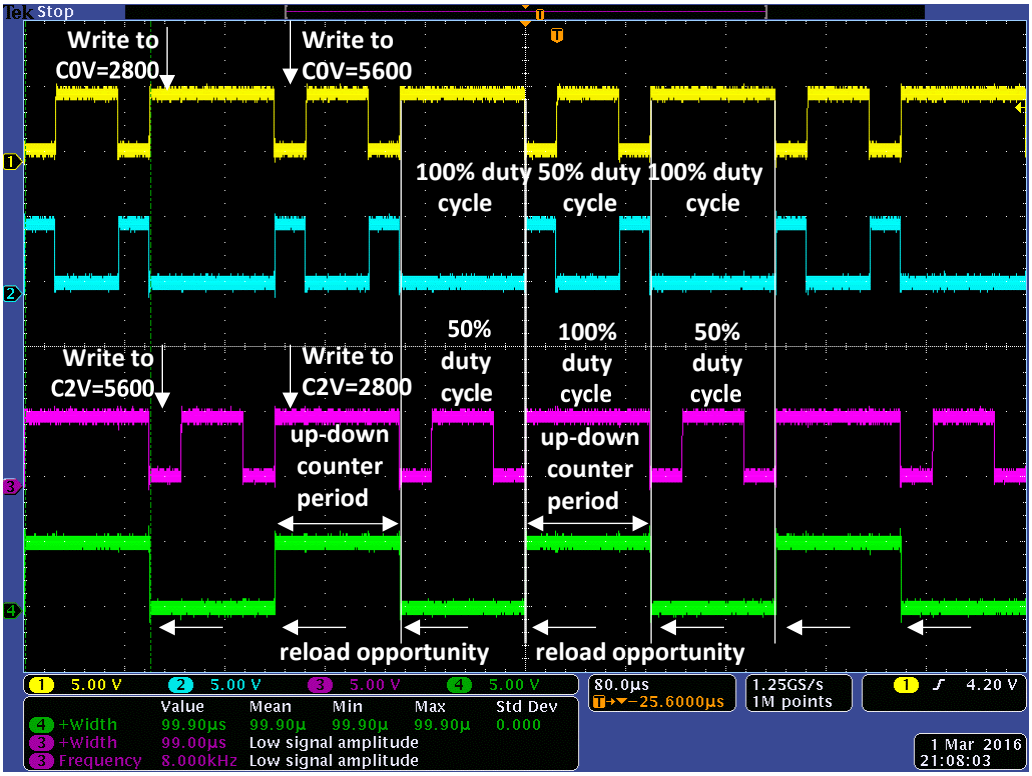

图13 . 中心对齐PWM和全周期重载策略

在图 12 和图 13 中，示波器通道 CH1、CH2 和 CH3 分别代表 FTM 模块的 FTM_CH0、FTM_CH1 和 FTM_CH2。FTM_CH0 和 FTM_CH1 工作在互补模式，与 FTM_CH2 和 FTM_CH3 相同。CH4 用于显示 FTM 寄存器的重新加载机会，该机会出现在加减计数器的每半个/全周期。在第一次重新加载机会中，100% 和 50% 占空比分别应用于互补通道对 FTM_CH0/CH1 和 FTM_CH2/CH3。下一次重加载机会时，C0V 和 C2V 寄存器从其新的缓冲值更新，因此 FTM_CH0/CH1 和 FTM_CH2/CH3 的占空比分别更改为 50% 和 100%。

4.3.3. 通过软件触发更新FTM寄存器——软件同步

所有双缓冲 FTM 寄存器也可以通过软件同步更新，使能 FTM_SYNC 寄存器中的 SWSYNC 位（如表 3 所示）。根据边界循环和所选计数模式的加载点，更新可以立即进行，也可以在所选的下一个加载点进行。

在向上计数模式下，边界周期定义为计数器返回到其初始值 (CNTIN) 的时间。在递增/递减计数模式下，边界周期定义为计数器从递减计数变为递增计数以及从递增计数变为递减计数的时间。在递增计数模式下，如果 CNTMIN 或 CTMAX 位之一为 1，则下一个加载点被使能。

如果 FTM_SYNCONF 寄存器中的 SWRSTCNT 位被置位，FTM 计数器会从 FTM_CNTIN 重新启动，并且 FTM 寄存器会立即更新。如果 SWRSTCNT 位清零，FTM 计数器保持计数，并且 FTM 寄存器在下一个加载点更新。

执行以下步骤来配置FTM模块为中心对齐PWM模式并使用软件同步更新FTM_CnV和FTM_OUTMASK 寄存器：

1. 初始化FTM0为中心对齐PWM模式，使能FTM_MODE寄存器中的FTMEN位和FTM_SC寄存器中的RIE位。
2. 使能SYNCONF寄存器中的SYNCMODE、SWWRBUF和SWOM位来选择CnV和OUTMASK寄存器的软件同步。
3. 通过FTM_SYNC[SWSYNC] = 1使能软件触发，等待下一个加载点。
4. 在重载机会中断例程中，更改CnV和OUTMASK寄存器的值并通过SWSYNC使能软件触发。

以下代码示例显示了中心对齐 PWM 模式下的 FTM0 初始化的一部分，并演示了用于更新 FTM_CnV 和 FTM_OUTMASK 寄存器的软件同步。

示例9 . 中心对齐PWM的软件同步

```
void CPWM_SoftSync_Init()
{
    ...

  /* 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 FTM_SYNCONF_SWRSTCNT_MASK;
    /* Enable PWM generation */
    FTM0->SC |= FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN2_MASK | FTM_SC_PWMEN3_MASK;
}

/* Interrupt routine for updating FTM registers - Software synchronization */
void FTM0_IRQHandler()
{
      if(flag)
      {
            /* Enable FTM0_CH0/FTM0_CH1 mask */
            FTM0->OUTMASK = FTM_OUTMASK_CH0OM_MASK |
            FTM_OUTMASK_CH1OM_MASK; FTM0-
            >CONTROLS[0].CnV=FTM_CnV_VAL(1120);
            FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(1120);
            if(flagMask)
            {
                  /* 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);
            }

           flagMask = !flagMask;
            }
       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(2800);
            FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(2800);
      }
           flag = !flag;
           PTD->PTOR |= 1<<3;                          // Toggle PTD3 to show reload opportunities FTM0-
           >SYNC |= FTM_SYNC_SWSYNC_MASK;              // Software sync
           FTM0->SC &= ~FTM_SC_RF_MASK;                // Clear Reload Flag bit
      }
```

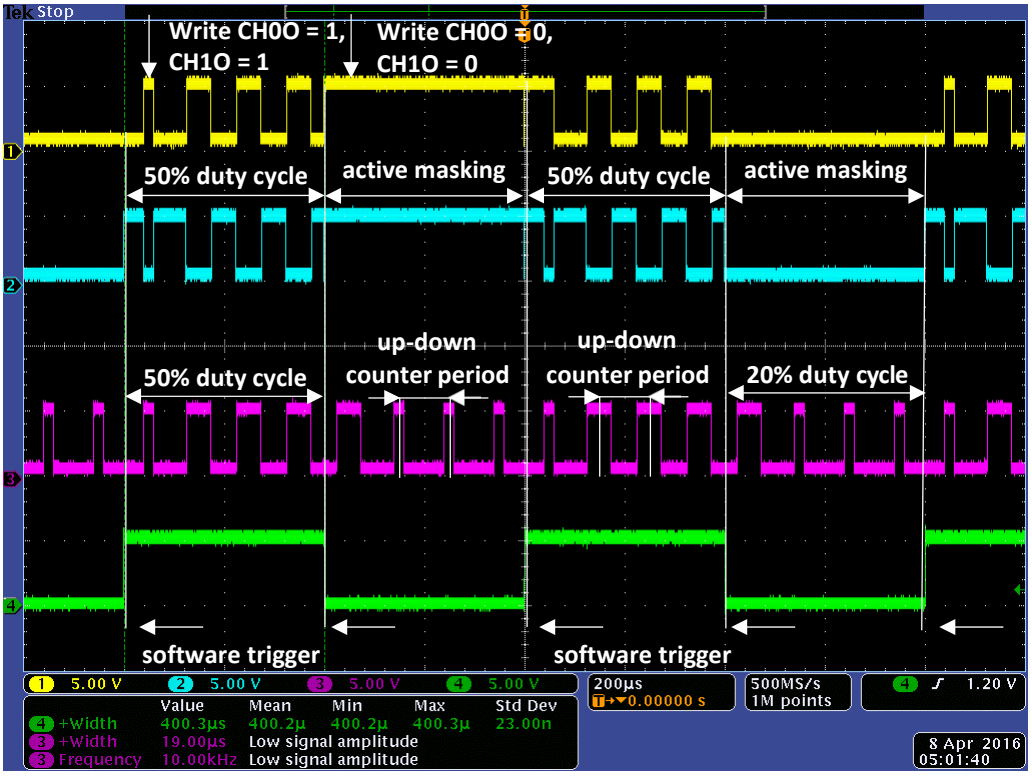

图14 . 中心对齐PWM的软件同步

在图 14 中，示波器通道 CH1、CH2 和 CH3分别代表FTM模块的FTM_CH0、FTM_CH1 和FTM_CH2。FTM_CH0和FTM_CH1工作在互补模式，与FTM_CH2和FTM_CH3相同。通道 CH4 用于显示每四个 PWM 周期发生一次的软件触发。通道对 FTM_CH0/FTM_CH1 和 FTM_CH2/FTM_CH3 的占空比从 50% 变为 20%（反之亦然），具体取决于更新 C0V和C2V 寄存器的软件触发事件。此外，第一个互补通道对 FTM_CH0/FTM_CH1显示了掩蔽特征。

4.3.4. 通过硬件触发更新FTM寄存器——硬件同步

硬件同步是在计数器运行时更新 FTM 寄存器的另一种方式。在这种情况下，中断服务不是必需的，因为 FTM 寄存器可以在代码执行期间随时更改，并且一旦发生硬件触发，它们的值就会更新。通过这种方式可以显着降低 CPU 负载。

根据 FTM_SYNC 寄存器中使能的 TRIGn 位，可以选择 FTM 模块的三个硬件触发信号输入。如果设置了 TRIG0，则硬件触发源来自许多其他模块，例如 LPIT 和 CMP（以及其他），可通过灵活的 TRIGMUX 模块来配置。通过设置 FTM_SYNC 寄存器中的 TRIG1 位，根据 SIM_FTMOPT1 寄存器中的 FTM0SYNCBIT 位生成硬件触发。在本节中，通过使能 TRIG2 位来演示第三个选项。在这种情况下，FTMx_FLT0 故障引脚被选为 FTM 硬件同步的硬件触发输入。

1. 初始化FTM0为中心对齐PWM模式并使能FTM_MODE寄存器中的FTMEN位。
2. 使能FTM0_SYNCONF寄存器中的SYNCMODE和HWWRBUF位来选择硬件同步。
3. 使能FTM0_SYNCONF寄存器中的TRIG2位，选择FTM0_FLT0故障引脚作为FTM0硬件同步的硬件触发输入。
4. 初始化FTM1模块，为FTM0硬件触发输入产生一个周期信号。为了更好地演示 FTM0 硬件同步，FTM1 的频率必须低于 FTM0 的频率。
5. 通过使能 FTM1_SC 寄存器中的 RIE 位并更改 C0V 和 C2V 寄存器中的值来生成重新加载机会中断。等待更新缓冲区值的硬件触发事件

示例10 . 中心对齐PWM的硬件同步

```
void CPWM_HardSync_Init()
{
	/* Enable clock for PORTA */
	PCC->PCCn[PCC_PORTA_INDEX] = PCC_PCCn_CGC_MASK;
	/* Set PTD14 as fault pin of FTM0 */ PORTA-
	>PCR[14] = PORT_PCR_MUX(2);

	...

	/* 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_PWMEN2_MASK | FTM_SC_PWMEN3_MASK;
}

/* Set FTM1 as a hardware trigger source for FTM0 through FTM0_FLT0 input PTB2 and PTA14
    have to be interconnected to each other */
void FTM1_Init()
{
	/* Enable clock for FTM1 */
	PCC->PCCn[PCC_FLEXTMR1_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;
	/* Enable clock for PORTB */
	PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;

	PORTB->PCR[2] = PORT_PCR_MUX(2);                    // PTB2 is used for FTM1-Channel0

	FTM1->SC=FTM_SC_CPWMS_MASK;                         // Select up-down counter for Center-Align PWM FTM1-
	>CONTROLS[0].CnSC=FTM_CnSC_ELSB_MASK;               // Select high-true pulses
	FTM1->MOD = FTM_MOD_MOD(15000-1);                   // Set modulo
	FTM1->CONTROLS[0].CnV=FTM_CnV_VAL(7500);            // Set channel Value
	FTM1->CNT = 0;                                      // Counter reset
	/* Select clock, enable reload opportunity interrupt and PWM generation */ FTM1-
	>SC|=FTM_SC_CLKS(1)|FTM_SC_RIE_MASK|FTM_SC_PWMEN0_MASK;
	/* Reload opportunity interrupt occurs when FTM counter reach CNTMAX value */ FTM1->SYNC |=
	FTM_SYNC_CNTMAX_MASK;
}

/* FTM1 interrupt routine to change values of FTM0 registers */
void FTM1_IRQHandler()
{
	if(flag)
	{
		FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(1000);
```

```
        FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(4000);
    }
    else
    {
        FTM0->CONTROLS[0].CnV=FTM_CnV_VAL(4000);
        FTM0->CONTROLS[2].CnV=FTM_CnV_VAL(1000);
    }
    flag = !flag;
    FTM1->SC &= ~FTM_SC_RF_MASK;          // Clear Reload Flag bit
}
```

代码运行的结果如下图所示：

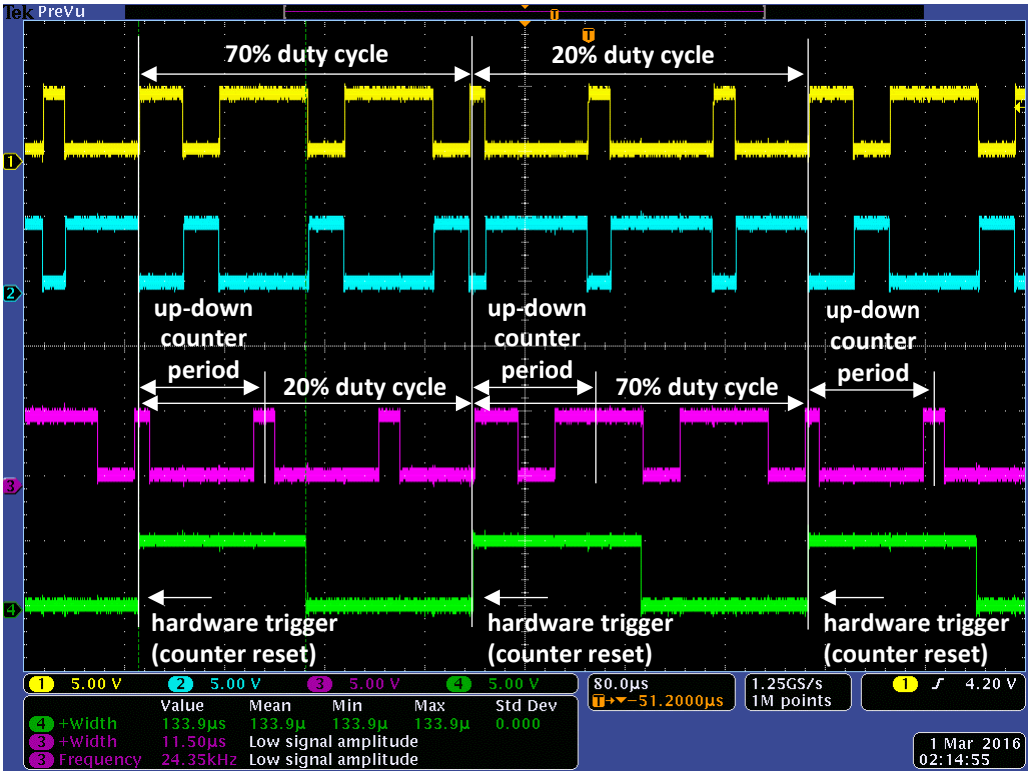

图15 . 中心对齐PWM的硬件同步

在图 15 中，示波器通道 CH1、CH2 和 CH3 分别代表 FTM0 模块通道 FTM0_CH0、FTM0_CH1 和 FTM0_CH2。FTM0_CH0 和 FTM0_CH1 工作在互补模式，与 FTM0_CH2 和 FTM0_CH3 相同。通道 CH4 显示由 FTM1 生成的硬件触发，其频率是 FTM0 频率的三分之一。接收到第一个硬件触发后，通道对 FTM_CH0/FTM_CH1 和通道对 FTM_CH2/FTM_CH3 的占空比分别从 20% 变为 70% 和从 70% 变为 20%。下一个硬件触发更新 C0V 和 C2V 寄存器的新值，相应通道对的占空比变回之前的值。

4.4. 全局时基 （GTB）

芯片上有多个FTM，但多个FTM模块是独立的。如果应用需要的 PWM 通道多于一个 FTM 所能提供的通道数，则可以使用多个 FTM 模块，但它们必须同步。两个（或多个）FTM 模块的同步意味着它们的计数器在任何时刻都具有相同的值。

S32K 设备提供了 GTB 机制来同步多个 FTM。GTB 是由主 FTM 生成的同步信号，它启动所有使用的 FTM 的计数器。使用 GTB 功能时必须满足这两个条件：每个 FTM 必须具有相同的时钟源，并且每个 FTM 必须同时启动。

要使能 GTB 功能，请对每个参与的 FTM 模块执行以下步骤：

1. 停止FTM计数器（将00b写入SC[CLKS]）。
2. 将FTM编程为预期配置。FTM 计数器模式必须在所有参与模块中保持一致。
3. 向CONF[GTBEEN]写入1，同时向CONF[GTBEOUT]写入0。
4. 在 SC[CLKS] 中选择预期的 FTM 计数器时钟源。 所有参与模块的时钟源必须一致。
5. 重置FTM计数器（向CNT寄存器写入一些值）

以下示例显示了FTM0 和 FTM2 的同步。两个 PWM 的占空比都调整为非常低的值，以更好地展示 GTB 机制的功能。

示例11 . 中心对齐PWM和GTB功能

```
void CPWM_and_Global_Time_Base()
{
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;

    /* Select and enable clock for FTM0 */
    PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;
    /* Select and enable clock for FTM2 */
    PCC->PCCn[PCC_FLEXTMR2_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

    PORTD->PCR[15] = PORT_PCR_MUX(2); // Set PTD15 for FTM0 - Channel0 PORTD->PCR[16] =
    PORT_PCR_MUX(2); // Set PTD16 for FTM0 - Channel1 PORTD->PCR[0] = PORT_PCR_MUX(4);
    // Set PTD0 for FTM2 - Channel0 PORTD->PCR[1] = PORT_PCR_MUX(4); // Set PTD1 for FTM2 -
    Channel1

    /* Select up-down counter for Center-Align PWM */ FTM0->SC =
    FTM_SC_CPWMS_MASK;
    FTM2->SC = FTM_SC_CPWMS_MASK;
    /* Enable registers updating from write buffers */ FTM0->MODE =
    FTM_MODE_FTMEN_MASK;
    FTM2->MODE = FTM_MODE_FTMEN_MASK;
    /* Enable complementary mode for channels pair n=1 */ FTM0-
    >COMBINE = FTM_COMBINE_COMP0_MASK;
    FTM2->COMBINE = FTM_COMBINE_COMP0_MASK;
    /* Set Modulo (10kHz PWM frequency @112MHz system clock) */ FTM0->MOD
    = FTM_MOD_MOD(5600-1);
    FTM2->MOD = FTM_MOD_MOD(5600-1);
    /* Set CNTIN */
    FTM0->CNTIN = FTM_CNTIN_INIT(0);
    FTM2->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;
    FTM2->CONTROLS[0].CnSC  =  FTM_CnSC_ELSB_MASK;
    FTM2->CONTROLS[1].CnSC =   FTM_CnSC_ELSB_MASK;
    /* Set Channel Value – 1% duty cycle */ FTM0-
    >CONTROLS[0].CnV=FTM_CnV_VAL(56); FTM2-
    >CONTROLS[0].CnV=FTM_CnV_VAL(56);
    /* FTM counter reset */ FTM0-
    >CNT = 0;
    FTM2->CNT = 0;

    /* Enable global time base to control FTM0 and FTM2 */ FTM0->CONF =
    FTM_CONF_GTBEEN_MASK;
    FTM2->CONF = FTM_CONF_GTBEEN_MASK;

    /* Select clock */
    FTM0->SC |= FTM_SC_CLKS(1);
    FTM2->SC |= FTM_SC_CLKS(1);

    /* Synchronization signal for FTM0 and FTM2 */ FTM0-
    >CONF |= FTM_CONF_GTBEOUT_MASK;

    /* Enable PWM generation */
    FTM0->SC |= FTM_SC_PWMEN0_MASK |
    FTM_SC_PWMEN1_MASK; FTM2->SC |=
    FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK;
}
```

未激活 GTB 和激活 GTB 的 FTM0 和 FTM2 的中心对齐 PWM 如下两图所示：

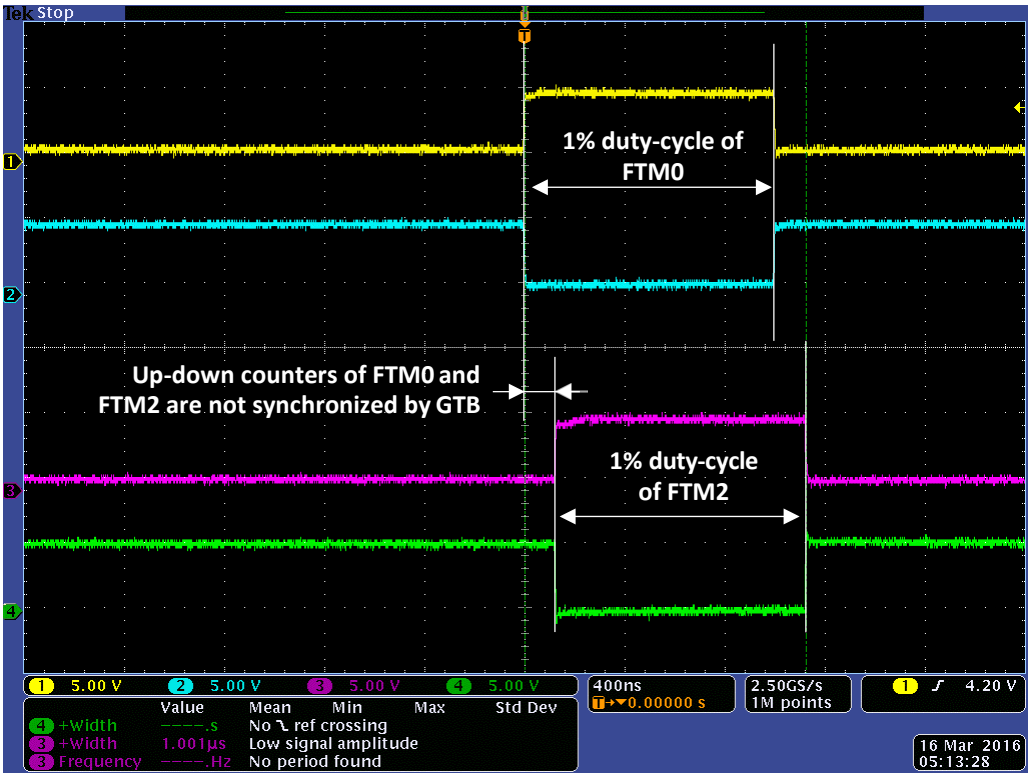

图16 . 未激活GTB的中心对齐PWM

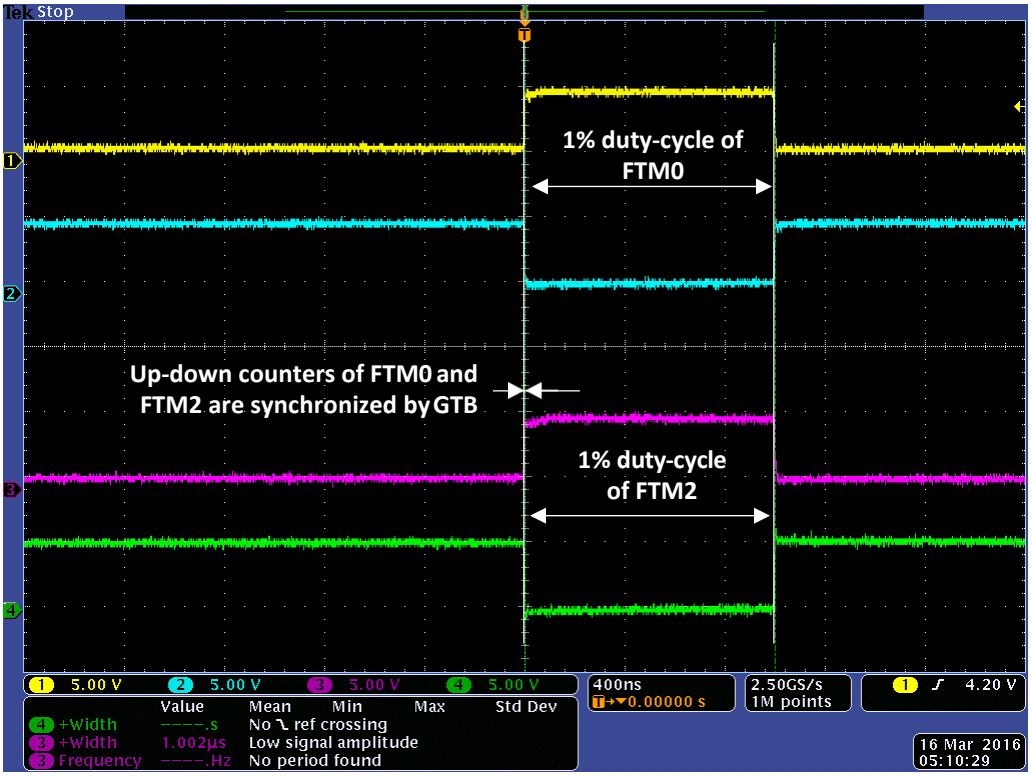

图17 . 激活GTB的中心对齐PWM

在图 16 和图 17 中，示波器通道 CH1、CH2、CH3 和 CH4 分别代表 FTM0 和 FTM2 模块的通道 FTM0_CH0、FTM0_CH1、FTM2_CH0 和 FTM2_CH1。 FTM0_CH0 和 FTM0_CH1 工作在互补模式，与 FTM2_CH0 和 FTM2_CH1 相同。调整示波器的时基，以便在示波器显示屏上可以清楚地看到通道 FTM0 和 FTM2 的 1% 占空比。如果全局时基未激活，则所使用的 FTM 的加减计数器不会完全对齐，它们的通道信号也不会对齐（参见图 16）。通过将 GTB 功能应用于 FTM 模块来处理此类问题（参见图 17）。

注意

GTB 功能不提供 FTM 计数器的连续同步，这意味着 FTM 计数器可能会在 FTM 操作期间失去同步。GTB 功能仅允许 FTM 计数器启动时同步。

4.5. ADC由FTM和PDB模块触发

在电机控制应用以及开关电源应用中，有必要测量 PWM 信号中心的电流。功率器件在这个瞬间没有切换，因此分流电阻上检测到的压降是稳定的。

S32K 设备的所有四个 FTM 模块都有多达八个通道。六个通道通常足以生成控制常用三相电机所需的 PWM 信号。FTM 模块的其余两个通道随后可用于控制 ADC 转换的时间点。

FTM 模块的触发信号可以在 FTM_EXTTRIG 寄存器中选择。所有 FTM 通道均可用于触发 ADC 模块。例如，如果在 FTM_EXTTRIG 寄存器中设置 CH0，则 FTM 计数器计数，直到达到写入 C0V 寄存器中的值。此时，CH0 产生一个信号，在每个 FTM 计数器周期触发 ADC 模块。如果 FTM_EXTTRIG 寄存器中的 INITTRIG 位被设置，则每次 FTM 计数器达到 CNTIN 寄存器的值时都会产生触发信号。

ADC 模块的默认和建议硬件触发方案是通过可编程延迟块 (PDB)。S32K配备了两个成对工作的 ADC 模块和两个 PDB 模块。这意味着 PDB0 与 ADC0 链接，PDB1 与 ADC1 链接。每个 PDB 模块都可以由软件或硬件触发。FTM 模块是可以通过 TRGMUX 寄存器 TRGMUX_PDB0 和 TRGMUX_PDB1 选择的触发源之一。

要在中心对齐 PWM模式下 并通过 FTM0 到 PDB0 触发 ADC0，请执行以下步骤：

1. 初始化FTM0模块为中心对齐PWM模式并使能FTM0_EXTTRIG寄存器中的 INITTRIGEN 位产生ADC0模块的硬件触发信号。
2. 通过向TRGMUX_PDB0寄存器中的SEL0位域写入0x16，配置TRGMUX模块由 FTM0通过PDB0模块触发ADC0。
3. 通过将适当的值写入 PDB0_MOD 和 PDB0_CH0DLY0 寄存器，延迟 PDB0 模块的 FTM0 初始化触发，以在 PWM 周期的中心触发 ADC0。
4. 初始化ADC0模块，使能ADC0_SC2寄存器中的ADTRG位，由FTM0模块触发 ADC0。

以下代码示例演示了 FTM0、PDB0 和 ADC0 模块的配置，其中 FTM0 用于生成中心对齐 PWM信号并通过 PDB0 模块触发 ADC0。

示例12 . ADC由FTM和PDB模块触发

```
/* FTM0 Initialization */
void FTM0_Init()
{
    /* Enable clock for PORTD */
    PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
    /* Select and enable clock for FTM0 */
    PCC->PCCn[PCC_FLEXTMR0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK;

    PORTD->PCR[15] = PORT_PCR_MUX(2); // Set PTD15 for FTM0 –Channel0 PORTD-
    >PCR[16] = PORT_PCR_MUX(2); // Set PTD16 for FTM0 –Channel1 PORTD->PCR[0] =
    PORT_PCR_MUX(2);  // Set PTD0 for FTM0 – Channel2  PORTD->PCR[1]  =
    PORT_PCR_MUX(2);  // Set PTD1 for FTM0 – Channel3  PORTD->PCR[2]  =
    PORT_PCR_MUX(1); // Set PTD2 as aPTD2
    PORTD->PCR[3] = PORT_PCR_MUX(1);  // Set PTD3  as aPTD3
    PTD->PDDR = (1<<2) | (1<<3);                  // Set PTD2 and PTD3 as an output

    /* Select up-down counter for Center-Align PWM */ FTM0->SC =
    FTM_SC_CPWMS_MASK;
    /* Enable combine, complementary mode and dead-time for channels pair CH0/CH1 and CH2/CH3 */ FTM0->COMBINE =
    FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_DTEN0_MASK; FTM0-
    >COMBINE |= FTM_COMBINE_SYNCEN1_MASK | FTM_COMBINE_COMP1_MASK | FTM_COMBINE_DTEN1_MASK;
    /* Set Modulo (10kHz PWM frequency @112MHz system clock) */ FTM0->MOD
    = FTM_MOD_MOD(5600-1);
    /* 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(2800); // 50% duty cycle FTM0-
        >CONTROLS[1].CnV=FTM_CnV_VAL(2800);  // 50% duty cycle FTM0-
        >CONTROLS[2].CnV=FTM_CnV_VAL(2800);  // 50% duty cycle FTM0-
        >CONTROLS[3].CnV=FTM_CnV_VAL(2800); // 50% duty cycle
        /* FTM counter reset */ FTM0-
        >CNT = 0;
        /* Insert DeadTime (1us) */
        FTM0->DEADTIME = FTM_DEADTIME_DTPS(3) | FTM_DEADTIME_DTVAL(7);
        /* 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 |
                    FTM_SC_PWMEN2_MASK| FTM_SC_PWMEN3_MASK;
}
/* PDB0 Initialization */
void PDB0_Init()
{
        PCC->PCCn[PCC_PDB0_INDEX] = PCC_PCCn_CGC_MASK;                  // Enable clock for PDB0
        FSL_NVIC->ISER[PDB0_IRQn / 32] |= (1 << (PDB0_IRQn % 32));      // Enable interrupt PDB0->MOD
        = 11200;                                                        // Set Modulo
        PDB0->CH[0].C1 = PDB_C1_TOS(1) | PDB_C1_EN(1);                  // Select and enable Channel0 PDB0-
        >CH[0].DLY[0] = 5600;                                           // 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;
}
/* TRGMUX Initialization */
void TRGMUX_Init()
{
        PCC->PCCn[PCC_TRGMUX_INDEX] = PCC_PCCn_CGC_MASK;                // Enable clock for TRGMUX module
        /* Set FTM as a trigger source for PDB0 */
        TRGMUX->TRGMUXn[TRGMUX_PDB0_INDEX] = TRGMUX_TRGMUXn_SEL0(0x16);
}
void ADC0_Init()
{
        PCC->PCCn[PCC_ADC0_INDEX] = 0;                                  // Disable clock for ADC0 PCC-
        >PCCn[PCC_ADC0_INDEX] = PCC_PCCn_PCS(6) | PCC_PCCn_CGC_MASK; // Enable clock for ADC0 FSL_NVIC-
        >ISER[ADC0_IRQn / 32] |= (1 << (ADC0_IRQn % 32));               // Enable interrupt

        /* 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);
}
/* ADC0 interrupt routine */
void ADC0_IRQHandler()
{
        PTD->PTOR |= 1<<3;                          // Toggle PTD2 to show completed ADC conversions
        ADC0_RA = ADC0->R[0];                       // Read ADC result register
}
/* PDB0 interrupt routine */
void PDB0_IRQHandler()
{
        PTD->PTOR |= 1<<2;                          // Toggle PTD2 to show FTM0 trigger signal PDB0->SC
        &= ~PDB_SC_PDBIF_MASK;                      // Clear PDB interrupt flag
}
```

代码示例的运行结果如下图所示：

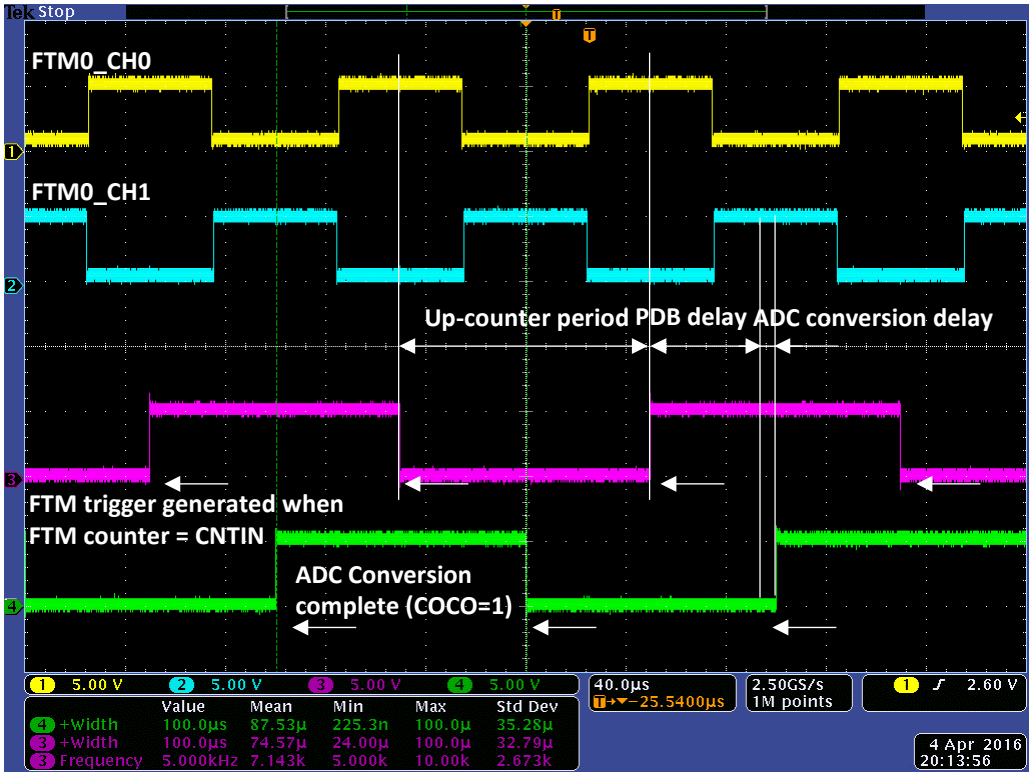

图18 . ADC由FTM和PDB模块触发

在图 18 中，示波器通道 CH1 和 CH2 分别代表 FTM0 通道 FTM0_CH0 和 FTM0_CH1。通道对 FTM0_CH0/FTM0_CH1 工作在互补模式，与通道对 FTM0_CH2/FTM0_CH3 相同。示波器通道 CH3 和 CH4 演示了 FTM0、PDB0 和 ADC0 模块之间的互连。通道 CH3 表示每次 FTM0 计数器达到 CNTIN 寄存器的值时 PDB0 中断程序中让PTD2 引脚翻转，每次 ADC0 转换完成（COCO = 1）时，通道 CH4 显示 ADC0 中断程序中让 PTD3 引脚翻转。FTM0 触发信号由 PDB0 模块延迟，以便 ADC0 转换发生在 PWM 信号的中心。

5. 修订历史

下表总结了自初始发布以来对文档所做的更改：

表5 . 修订历史

Revision number	Date	Substantive changes
0	06/2016	Initial release.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN5303
Rev. 0
06/2016