Application Brief

如何调试中断异常

Ryan Ma, Shaoxing Ke

引言

在 CPU 上支持实时任务需要使用中断。如果外部传感器检测到故障，则需要中断或停止 CPU，以执行能够处理故障的子例程。在此示例中，信号到达 CPU 时中断的时序至关重要。中断是硬件或软件驱动的信号，可导致 CPU 暂停当前的程序序列并执行子例程。中断通常处理对应用至关重要并需要及时执行的时间关键循环和控制算法。大多数情况下，中断能以已知频率定期发生。但是，在设计软件架构时，您是否曾看到中断波形出现错误振荡，如 图 1 中所示？

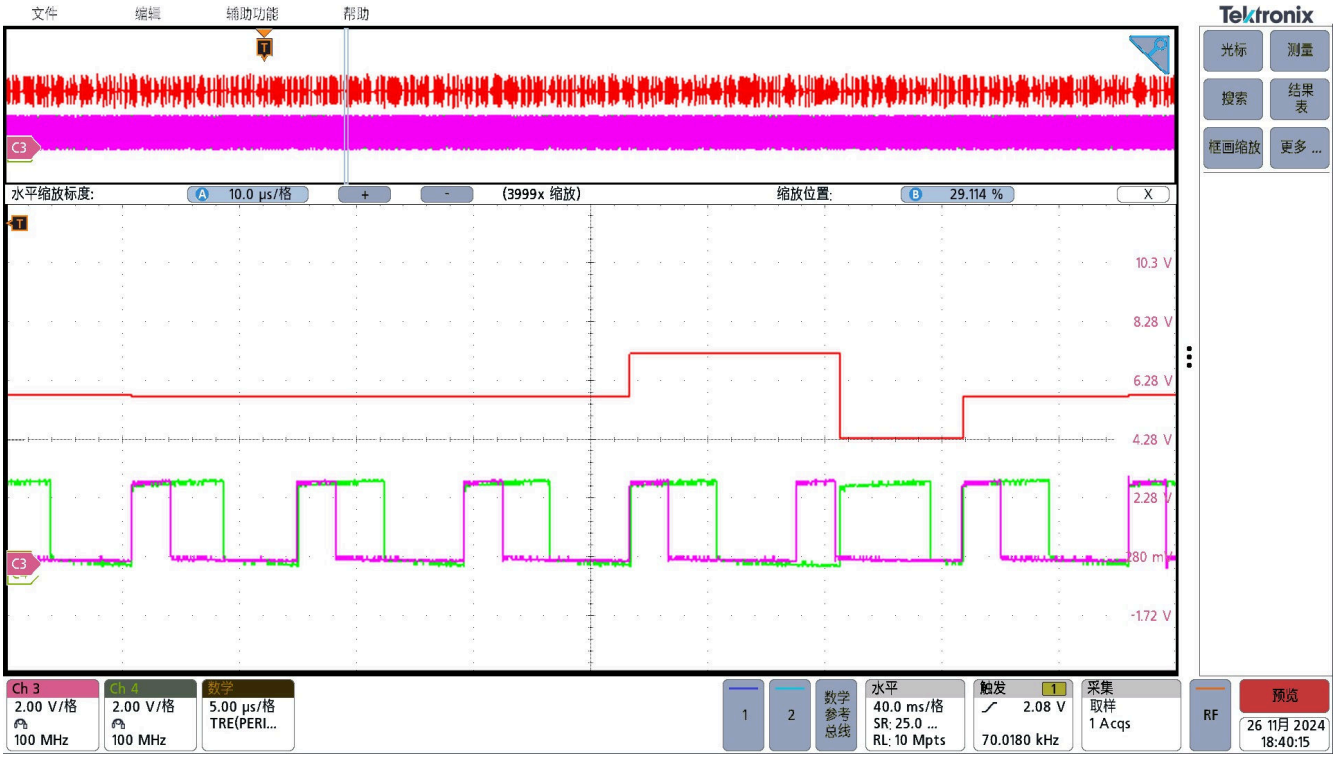

图 1. 异常中断振荡（CH4：通过 GPIO 切换中断；Ch3：发生 ePWM ZRO 事件时触发中断，红色信号：示波器的频率趋势测量）

中断传播路径和中断时序

首先，需要重点介绍中断延迟的两个概念，即中断传播路径和中断时序。中断传播路径是从触发中断请求到中断服务函数开始的时间。其次，确认在触发中断请求期间或在正常中断执行期间是否存在任何干扰因素。第三，通过合理设置中断优先级（例如中断嵌套和寄存器堆栈恢复/保护）并屏蔽其他中断干扰源，可以保持中断延迟正常执行。

C28x 上的中断传播路径主要分四个阶段处理中断：

1. 接收中断请求。如 图 2 中所述，必须通过软件中断（来自程序代码）或硬件中断（来自引脚或片上器件）请求才能暂停当前程序序列。

2. 批准中断。C28x 必须批准中断请求。如果中断可屏蔽，则必须满足某些条件，C28x 才能批准中断请求。对于不可屏蔽的硬件中断和软件中断，将立即批准，如 图 3 中所述。
3. 准备中断服务例程并保存寄存器值，如 图 4 中所述。
4. 执行中断服务例程。这是中断循环处理入口，调用 ISR。

大多数编程人员只关注前两个阶段，对后两个阶段的堆栈保护或恢复以及中断响应了解较少。本应用简报将深入探讨第三和第四阶段。

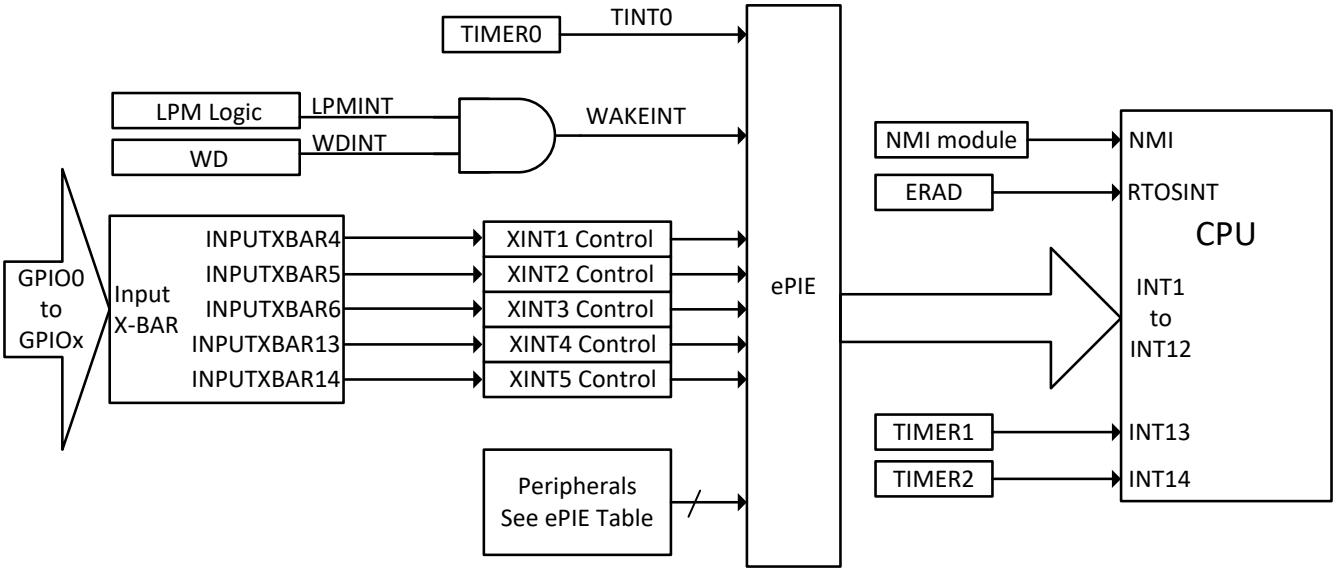

图 2. 中断触发源 (F28003x)

图 3 展示外设中断如何传播到 CPU。

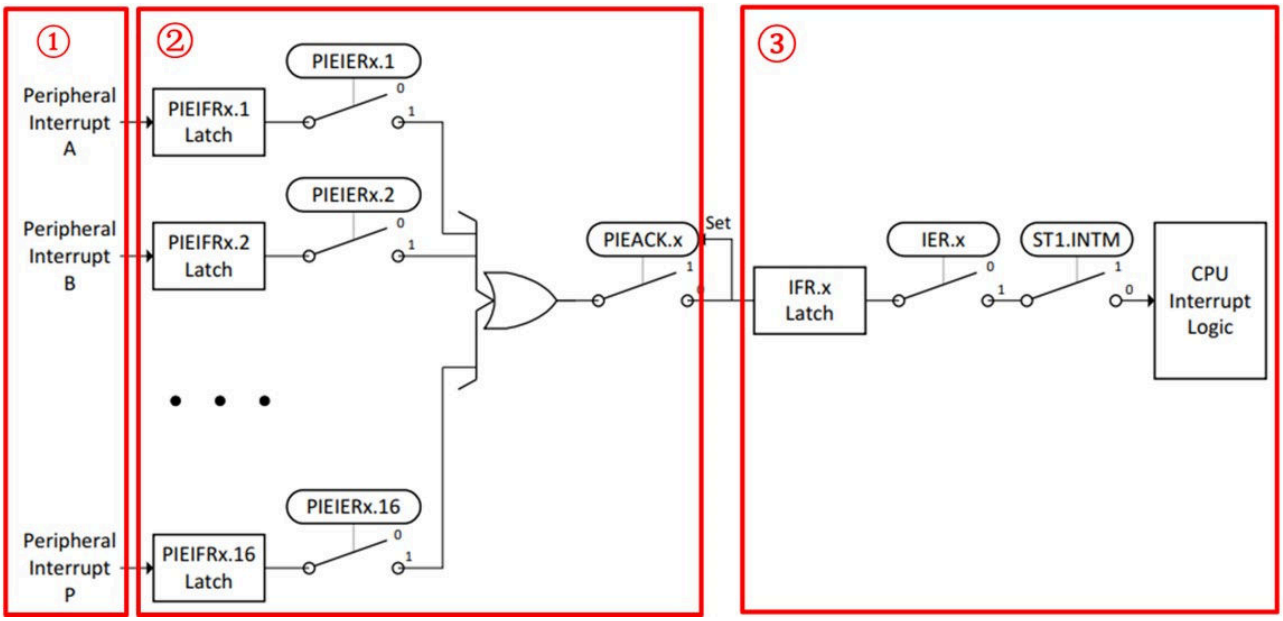

图 3. 中断传播路径

图 4 展示 C28x 如何生成和响应中断服务函数。

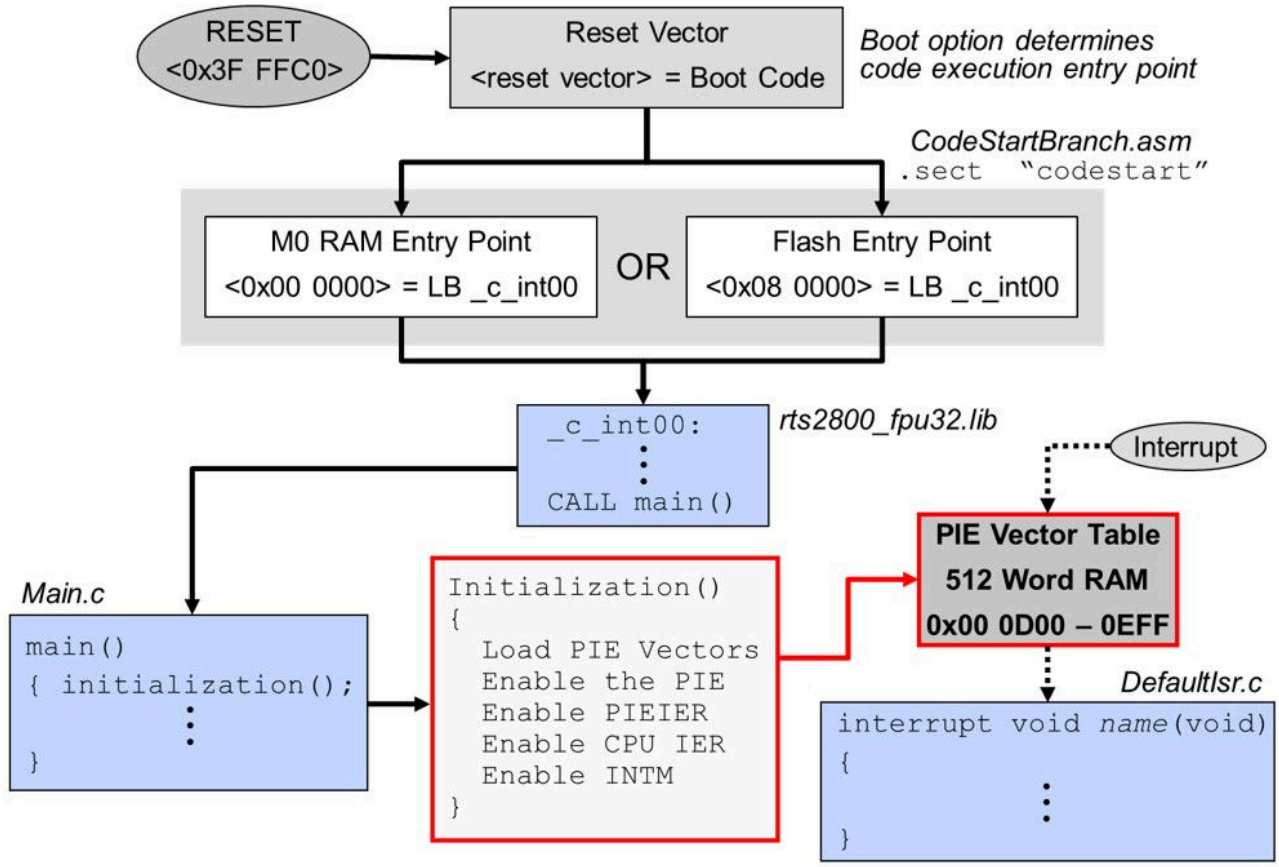

图 4. 中断 PIE 初始化代码流程

从中断请求触发到中断服务函数 ISR 的中断时序：

1. 最小延迟（到 ISR 中发生实际工作的时间），14 或 16 个周期：以 F280039C 120MHz CPU 为例，最小延迟（加上所有寄存器在准备实时中断时自动保存或恢复可能为 40 个周期）约为 50 个周期/415ns 延迟。
2. 最大延迟：取决于 C28x 处理堆栈保护和恢复、等待状态、INTM，以及不可中断 RPT 指令的周期，如 图 5 中所述。

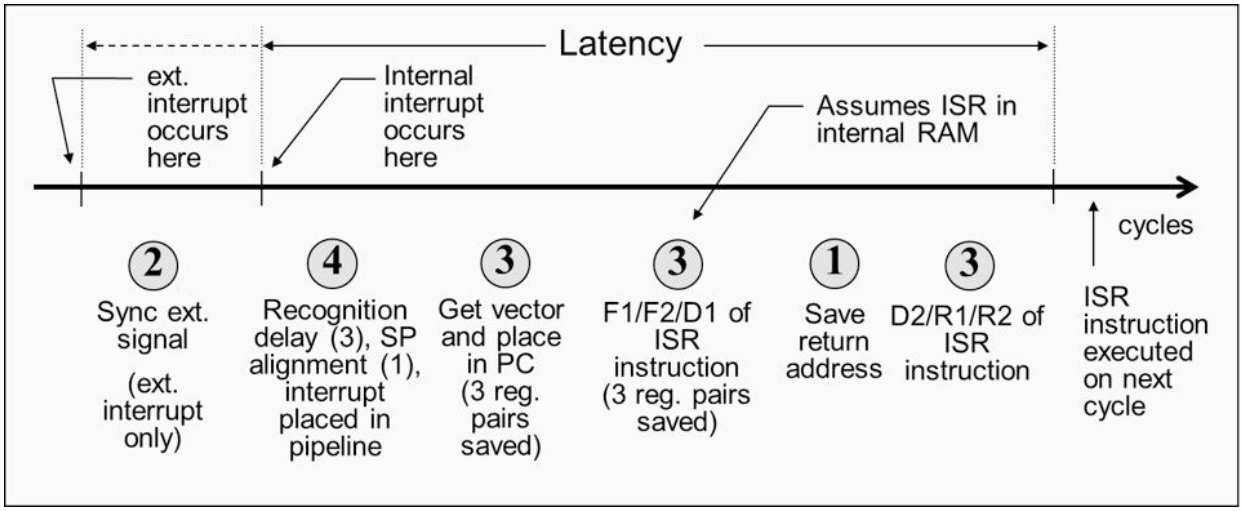

图 5. 中断延迟流程

除了正确使用中断请求和中断批准操作位（例如 INTM、IER 位），还要考虑以下可能影响中断的干扰因素和中断嵌套。

中断嵌套和干扰因素

- 当 C28x 正在执行中断或响应高优先级中断时，默认情况下，该中断无法继续响应其他低优先级中断。但是，可以通过一些步骤在当前中断内启用其他中断的服务。这称为中断嵌套。
- 当如 RPT 指令之类的不可中断指令执行时间过长或过于频繁时，C28x CPU 无法及时响应中断。

这两点是可能影响中断时序方式的源头。

中断嵌套

关于中断嵌套，中断通过 C28x 硬件自动设置优先级。可在特定器件系列专用的系统控制指南中找到所有中断的优先级。当 C28x CPU 响应低优先级中断时，CPU 会干扰高优先级中断的正常响应，如 图 6 中所述。

Table 3-3. Pie Channel Mapping

	INTx.1	INTx.2	INTx.3	INTx.4	INTx.5	INTx.6	INTx.7	INTx.8	INTx.9	INTx.10	INTx.11	INTx.12	INTx.13	INTx.14	INTx.15	INTx.16
INT1.y	ADCA1	ADCB1	ADCC1	XINT1	XINT2	-	TIMER0	WAKE / WDOG	-	SYS_ERR	-	-	-	-	-	-
INT2.y	EPWM1_TZ	EPWM2_TZ	EPWM3_TZ	EPWM4_TZ	EPWM5_TZ	EPWM6_TZ	EPWM7_TZ	EPWM8_TZ	-	-	-	-	-	-	-	-
INT3.y	EPWM1	EPWM2	EPWM3	EPWM4	EPWM5	EPWM6	EPWM7	EPWM8	-	-	-	-	-	-	-	-
INT4.y	ECAP1	ECAP2	ECAP3	-	-	-	-	-	-	-	ECAP3INT2	-	-	-	-	-
INT5.y	EQEP1	EQEP2	-	-	CLB1	CLB2	CLB3	CLB4	SDFM1	SDFM2	-	-	SDFM1DR1	SDFM1DR2	SDFM1DR3	SDFM1DR4
INT6.y	SPIA_RX	SPIA_TX	SPIB_RX	SPIB_TX	-	-	-	-	-	-	-	-	SDFM2DR1	SDFM2DR2	SDFM2DR3	SDFM2DR4
INT7.y	DMA_CH1	DMA_CH2	DMA_CH3	DMA_CH4	DMA_CH5	DMA_CH6	-	-	-	-	FSITX_INT1	FSITX_INT2	FSIRX_INT1	FSIRX_INT2	-	DCC0
INT8.y	I2CA	I2CA_FIFO	I2CB	I2CB_FIFO	-	-	-	-	LINA_0	LINA_1	LINB_0	LINB_1	PMBUSA	-	-	DCC1
INT9.y	SCIA_RX	SCIA_TX	SCIB_RX	SCIB_TX	DCANA_0	DCANA_1	-	-	MCAN_0	MCAN_1	MCAN_ECC	MCAN_WAKE	BGCRC_CPU	-	-	HICA
INT10.y	ADCA_EVT	ADCA2	ADCA3	ADCA4	ADCB_EVT	ADCB2	ADCB3	ADCB4	ADCC_EVT	ADCC2	ADCC3	ADCC4	-	-	-	-
INT11.y	CLA1_1	CLA1_2	CLA1_3	CLA1_4	CLA1_5	CLA1_6	CLA1_7	CLA1_8	-	-	-	-	-	-	-	-
INT12.y	XINT3	XINT4	XINT5	MPOST	FMC	-	FPU_OVERFLOW	FPU_UNDERFLOW	-	RAM_CORR_ERR	FLASH_CORR_ERR	RAM_ACC_VIOLATION	AES_SIN_TREQ	BGCRC_CLA1	CLA_OVERFLOW	CLA_UNDERFLOW

图 6. 中断 PIE 通道映射

因此，应用代码需要在低优先级中断期间添加简单的软件优先级。这使得 CPU 能够在执行低优先级中断时及时响应高优先级中断处理。以下是 C28x 执行中断嵌套的步骤：

1. 设置全局优先级：
   a. 修改 IER 寄存器，以允许为用户优先级较高的 CPU 中断提供服务。（注意：此时 IER 已保存在堆栈中。）
2. 设置组优先级：
   a. 修改相应的 PIEIERx 寄存器，以允许为用户设置优先级较高的组中断提供服务。（注意：除了此 ISR 服务的组以外，不要清除其他组的 PIEIER 寄存器位。这样做可能会导致发生错误的中断。）
3. 启用中断：执行此操作有三个步骤：
   a. 清除 PIEACK 位。
   b. 等待至少一个周期。
   c. 清除 INTM 位。使用汇编语句 asm(" CLRC INTM")；或者 TI 示例使用 #define EINT asm(" CLRC INTM")。
4. 运行 ISR 的主要部分。
5. 设置 INTM 以禁用中断。使用 asm(" SETC INTM")；或者 TI 示例使用 #define DINT asm(" SETC INTM")。
6. 恢复 PIEIERx（可选，具体取决于步骤 2）

7. 从 ISR 返回：
   a. 这将自动恢复 INTM 和 IER。同时，示例代码如下：

```
// // C28x ISR Code // // Enable nested interrupts // // ADCA1 interrupt for loop Interrput
void INT_myCPUTIMER2_ISR(void)
{
        uint16_t TempPIEIER;
        TempPIEIER = PieCtrlRegs.PIEIER1.all; // Save PIEIER register for later
        IER |= 0x001;                         // Set global priority by adjusting IER
        IER &= 0x001;
        PieCtrlRegs.PIEIER1.all &= 0x0001;    // Set group priority by adjusting PIEIER1
to //allow INT1.1 to interrupt current CPU time0 ISR
        PieCtrlRegs.PIEACK.all = 0xFFFF;      // Enable PIE interrupts
        asm("       NOP");                    // Wait one cycle
        EINT;                                 // Clear INTM to enable interrupts
        //
        // Insert ISR Code here.......
        // for now just insert a delay
        //
        //for(i = 1; i <= 10; i++) {}
        //
        // Restore registers saved:
        //
        DINT;
        PieCtrlRegs.PIEIER1.all = TempPIEIER;
}
```

我们的下一代 C29x 架构 F29H85x 支持硬件中断优先级，无需软件开销，并允许中断嵌套。与 C28x 的 40 个周期相比，C29x 架构的所有寄存器在实时中断时由硬件自动保存/恢复，仅需十个周期。

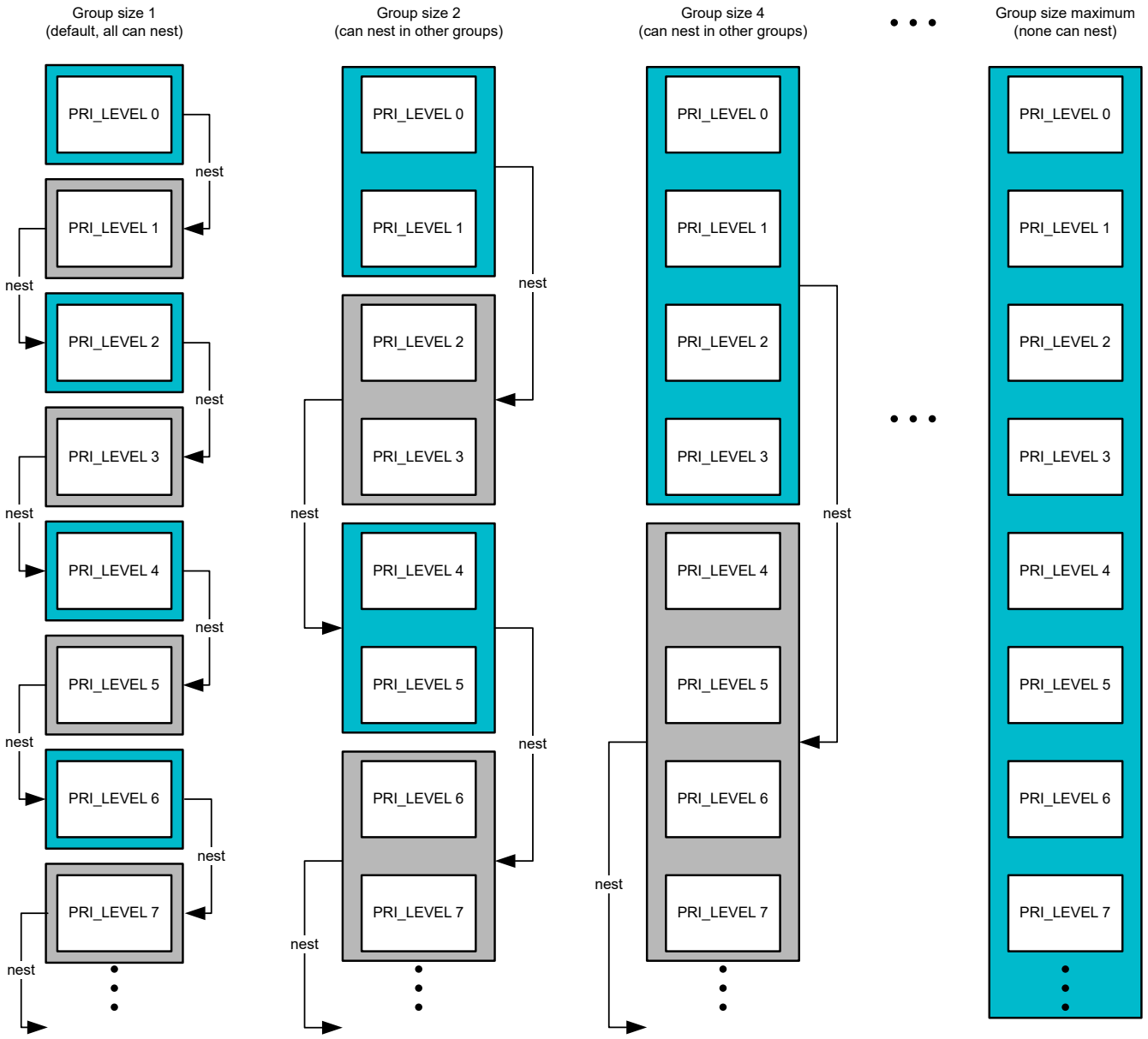

图 7. C29x 中断分组概述

在中断服务程序 (ISR) 中，可通过将 CPU 级别 DSTS.INTE 位设置为活动状态来启用 PIPE 模块中的 INT 嵌套，因为此位在进入 ISR 时禁用，如 图 7 中所述。以下是 C28x 执行中断嵌套的步骤：

```
// // C29x ISR Code // // Enable nested interrupts // // ADCA1 interrupt for loop Interrput
void INT_myCPUTIMER0_ISR(void)
{
       // Set INTE to 1 to enable interrupts here.
       ENINT;
       // Insert ISR Code here.......

}
```

C28x 中断嵌套测试结果

以下测试结果通过两个中断得出：150kHz 时的 EPWM 中断（黄色信号）和 1kHz 时的 Timer2 中断（蓝色信号）。Timer2 中断的优先级低于 EPWM 中断。如果未启用 C28x 中断嵌套，EPWM 的中断频率不会是 150kHz，如 图 8 中所示。只有利用 C28x CPU 中断嵌套，才能使 EPWM 中断固定在 150kHz，如 图 9 中所示。测试结果基于 LAUNCHXL-F280039C。如果未通过如上述代码所述的软件方法启用中断嵌套，则会出现异常中断行为。

启用中断嵌套后，即使发生了较低优先级的中断，仍然可以进入和执行较高优先级的中断。这可确保较高优先级的中断频率保持不变。

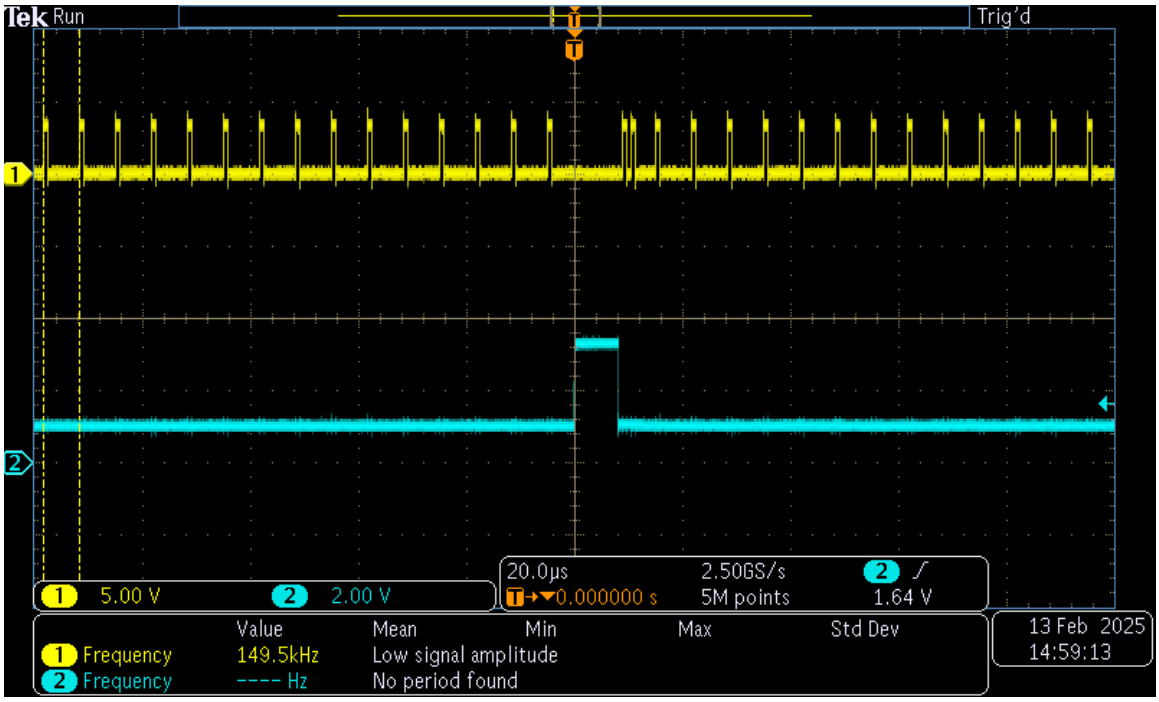

图 8. C28x 中断嵌套禁用测试结果（CH1：EPWM 中断，CH2：TIMER2 中断）

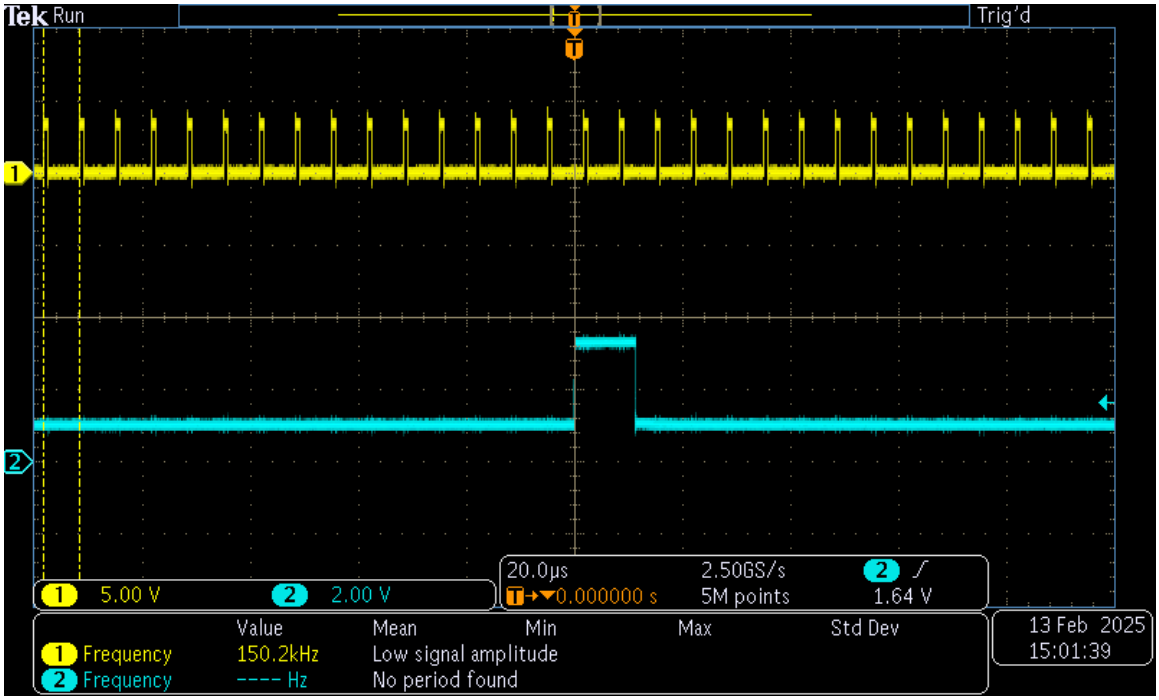

图 9. C28x 中断嵌套启用测试结果（CH1：EPWM 中断，CH2：TIMER2 中断）

C29x 中断嵌套测试结果

以下测试结果通过两个中断得出：150kHz 时的 EPWM 中断（粉色信号）和 1kHz 时的 Timer2 中断（绿色信号）。Timer2 中断的优先级低于 EPWM 中断。如果未启用 C29x 中断嵌套，EPWM 的中断频率不会是 150kHz，如 图 10 中所示。只有利用 C29x CPU 中断嵌套，才能使 EPWM 中断固定在 150kHz，如 图 11 中所示。此测试基于 F29x 器件。如果未通过如上述代码所述的软件方法启用中断嵌套，则会出现异常中断行为。

启用中断嵌套后，即使发生了较低优先级的中断，仍然可以进入和执行较高优先级的中断。这可确保较高优先级的中断频率保持不变。

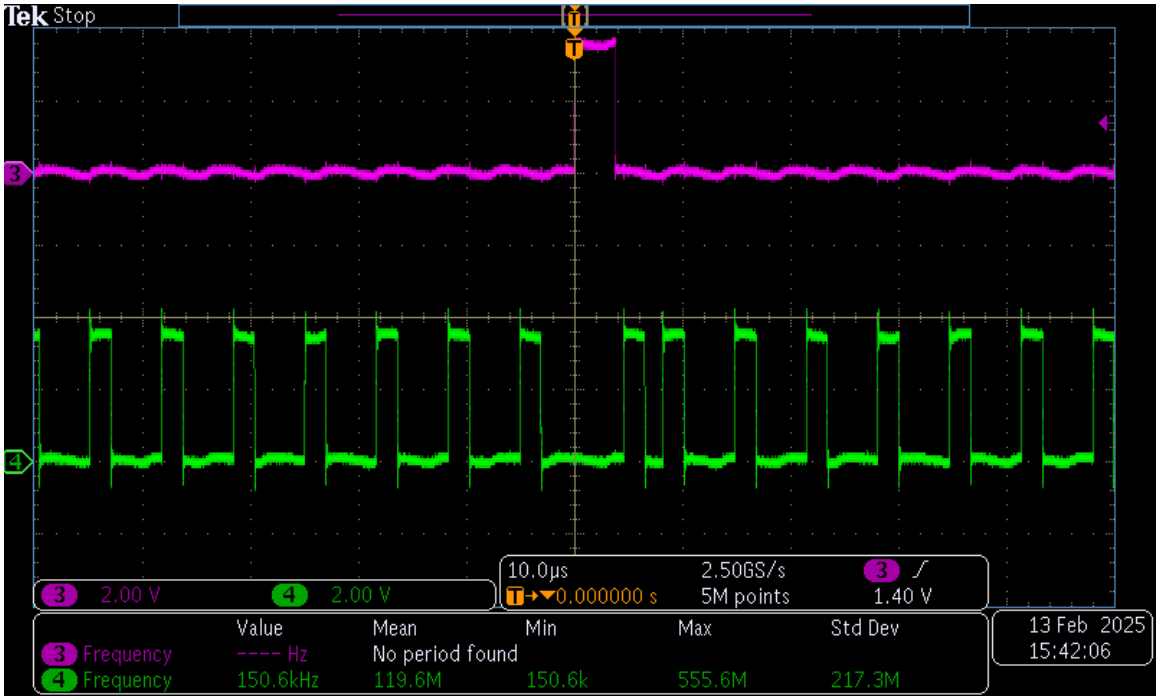

图 10. C29x 中断嵌套禁用测试结果（CH1：EPWM 中断，CH2：TIMER2 中断）

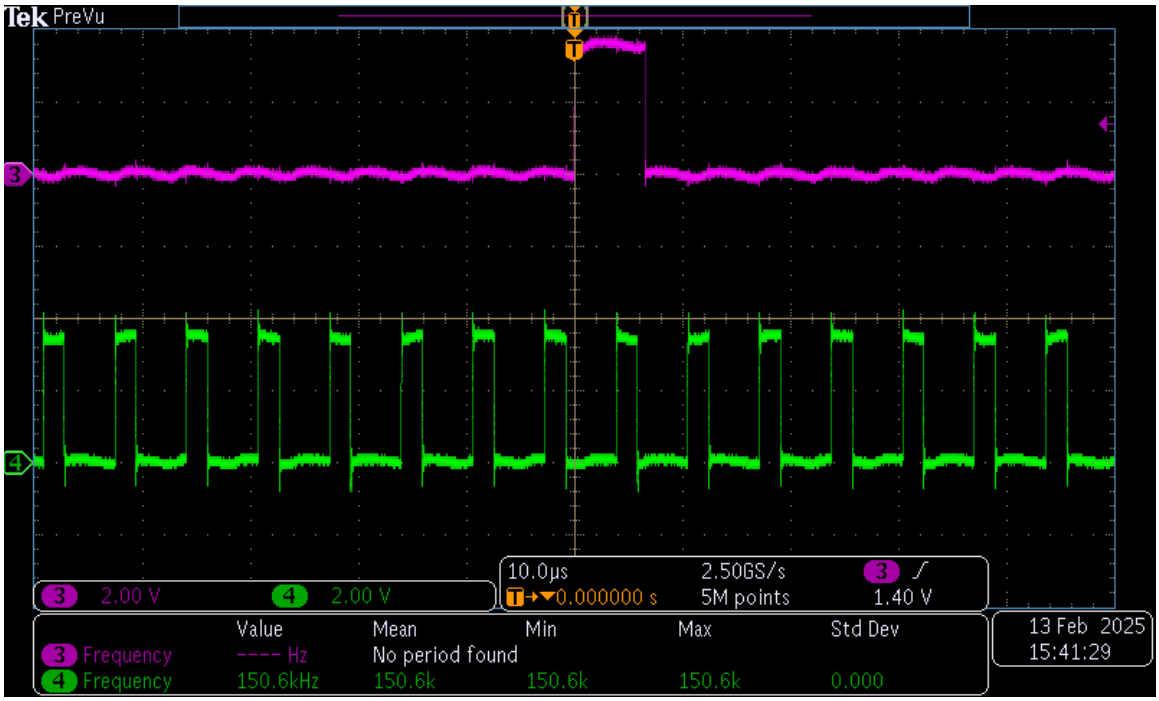

图 11. C29x 中断嵌套启用测试结果（CH1：EPWM 中断，CH2：TIMER2 中断）

不可中断指令影响中断时序

关于不可中断指令 RPT，如果在主程序或状态机中使用大量重复的全局初始化变量（例如 Memcopy、for 循环为相同数组赋值或执行重复操作），C2000 编译器会自动生成 RPT 指令。重复 (RPT) 指令允许单条指令执行 (N + 1) 次，其中 N 指定为 RPT 指令的操作数。该指令执行一次，然后重复 N 次。执行 RPT 时，重复计数器 (RPTC) 会加载 N。然后，每次执行重复指令时，RPTC 递减，直到 RPTC 等于 0。有关 RPT 的说明和可重复指令列表，请参阅 *TMS320C28x CPU 和指令集参考指南* 的 *C28x 汇编语言指令* 一章中的 *RPT *8bit/loc16* 部分。

由于这条 RPT 指令不可中断，它不具备上下文保存堆栈保护或恢复功能。因此，PC 指针此时留在 RPT 中，它可能无法及时响应中断请求，如 图 12 中所述。

RPT #8bit/loc16 ***Repeat Next Instruction***

Syntax Options

Syntax Options	Opcode	Objmode	RPT	CYC
RPT #8bit	1111 0110 CCCC CCCC	X	–	1
RPT loc16	1111 0111 LLLL LLLL	X	–	4

Operands

#8bit – 8-bit constant immediate value (0 to 255 range)

loc16 – Addressing mode (see Chapter 5)

Description

Repeat the next instruction. An internal repeat counter (RPTC) is loaded with a value N that is either the specified #8bit constant value or the content of the location pointed to by the "loc16" addressing mode. After the instruction that follows the RPT is executed once, it is repeated N times; that is, the instruction following the RPT executes N + 1 times. Because the RPTC cannot be saved during a context switch, repeat loops are regarded as multicycle instructions and are not interruptible.

Note on syntax: Parallel bars (||) before the repeated instruction are used as a reminder that the instruction is repeated and is not interruptable. When writing inline assembly, use the syntax

```
asm(|| RPT #8bt/ loc16 || instruction");
```

Not all instructions are repeatable. If an instruction that is not repeatable follows the RPT instruction, the RPTC counter is reset to 0 and the instruction only executes once. The 28x Assembly Language tools check for this condition and issue warnings.

Flags and Modes

None

Repeat

This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Example

```
; Copy the number of elements specified in VarA from Array1
; to Array2:
; int16 Array1[N]; // Located in high 64K of program space
; int16 Array2[N]; // Located in data space
; for(i=0; i < VarA; i++)
;    Array2[i] = Array1[i];
  MOVL XAR2,#Array2  ; XAR2 = pointer to Array2
  RPT @VarA          ; Repeat next instruction
                     ; [VarA] + 1 times
|| XPREAD            ; Array2[i] = Array1[i],
*XAR2++,*(Array1)    ; i++
```

图 12. RPT 指令简介

因此，您可以使用正确的设置进入 C2000 编译器，也可以避免生成 C 语言使用说明。关于 C2000 编译器，可以更改项目属性 -> C2000 编译器 -> 高级选项 -> 运行时模型选项 -> 启用“不生成 RPT 指令”，如 图 13 中所述。

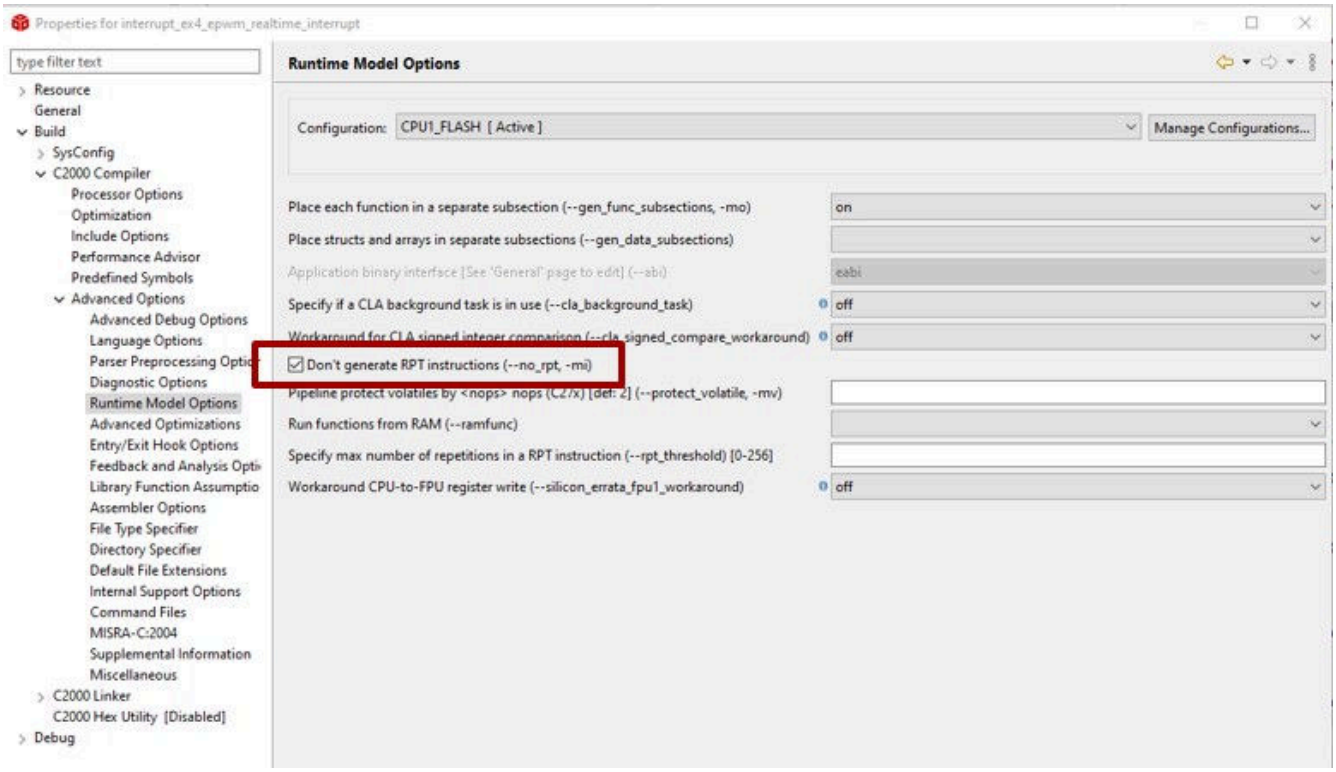

图 13. 关于 RPT 指令的 C2000 编译器设置

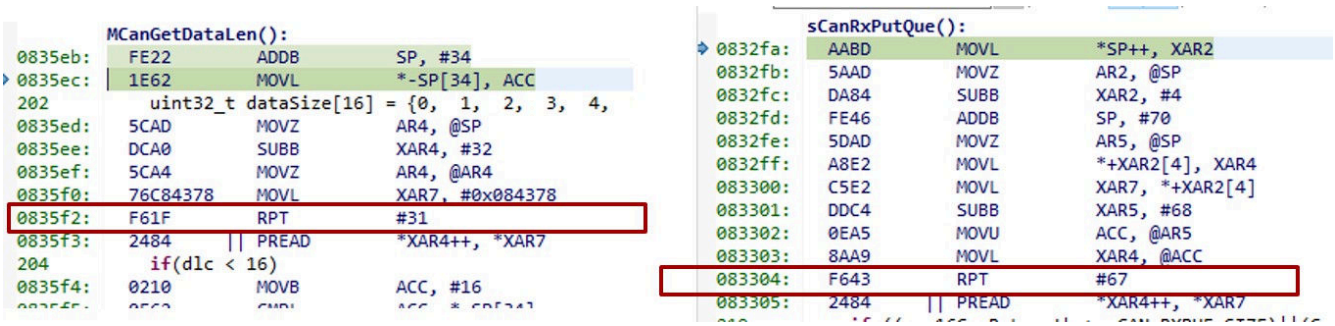

图 14. 由上述函数生成的 RPT 指令

```
202 uint32_t MCanGetDataLen(uint32_t dlc)
203 {
204     uint32_t dataSize[16] = {0,  1,  2,  3,  4,  5,  6, 7, 8,
205                              12, 16, 20, 24, 32, 48, 64};
206     if(dlc < 16)
207     {
208         return(dataSize[dlc]);
209     }
210     else
211     {
212         return 0;
213     }
214
215 }
```

```
217 void sCanRxPutQue(CANFRAME CanRx)
218 {
219     //uint16_t u16UDSRxDataFlag = 0;
220
221     if ((s_u16CanRxLength >= CAN_RXBUF_SIZE)||(CanRx.u8Len !=8))
222     {
223         return;
224     }
225
226     *s_pCanRxIn = CanRx;
227     if(s_pCanRxIn->CanId.all!= 0)
228     {
229
230         if (s_pCanRxIn == &s_aCanRxBuf[CAN_RXBUF_SIZE - 1])
231         {
232             s_pCanRxIn = s_aCanRxBuf;
233         }
234         else
235         {
236             s_pCanRxIn++;
237         }
238         s_u16CanRxLength++;
239     }
240 }
```

图 15. 源代码

最后，按照上述 C2000 编译器设置操作，EPWM ISR 将正常工作，如 图 16 中所述。

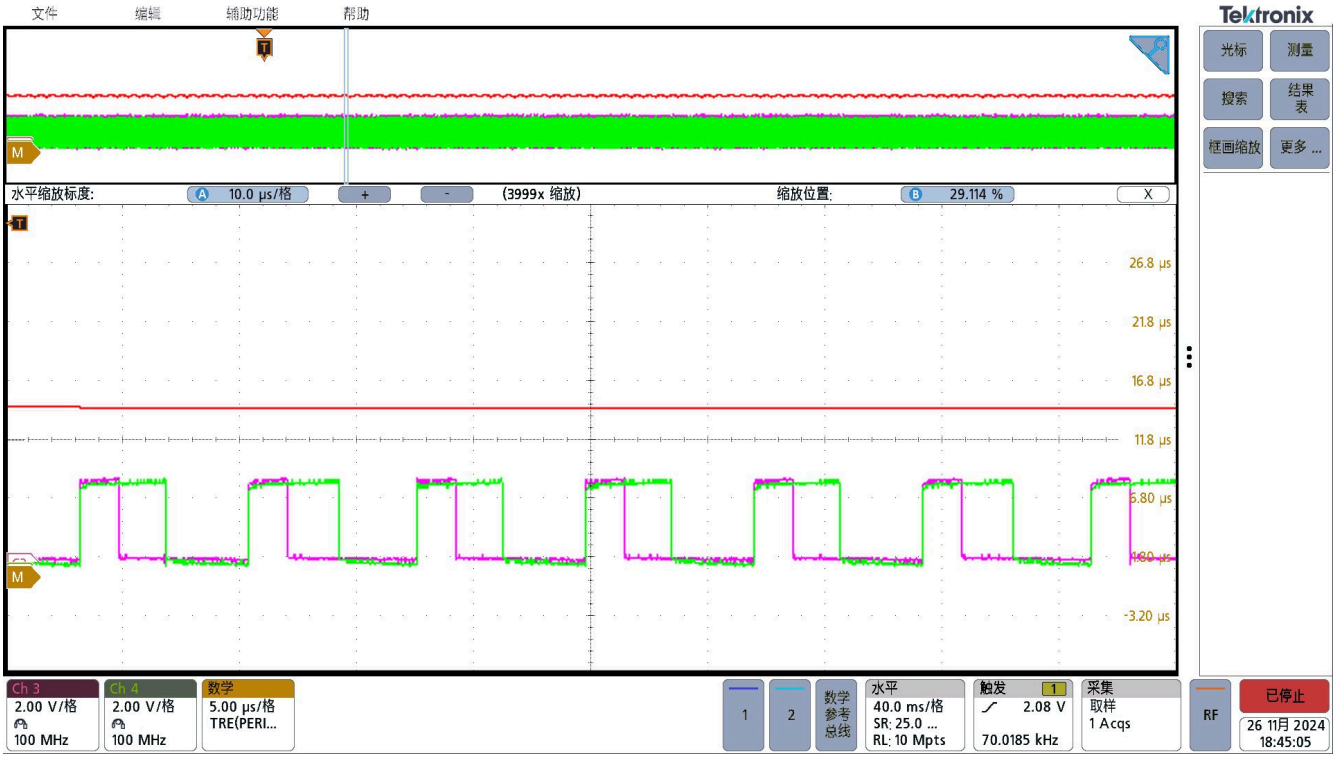

图 16. 异常中断振荡已修复（CH4：通过 GPIO 切换中断；Ch3：发生 ePWM ZRO 事件时触发中断，红色信号：示波器的频率趋势测量）

总结

通过遵守最小中断延迟，中断得以正常执行。然而，影响中断正常执行的因素包括：

- 中断请求是否正常触发
- 是否启用 INTM 或 IER 启用位，以及 IFG 标志是否正常设置
- 中断传播路径是否可能已阻止
- 保存寄存器时，中断响应是否可能受到干扰，如不可中断指令 RPT
- 无论中断是否嵌套，当低优先级中断响应时，高优先级中断会被阻止，从而影响中断响应的及时性

本技术文章介绍如何找到影响中断的因素并进行故障排除。

商标

所有商标均为其各自所有者的财产。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265