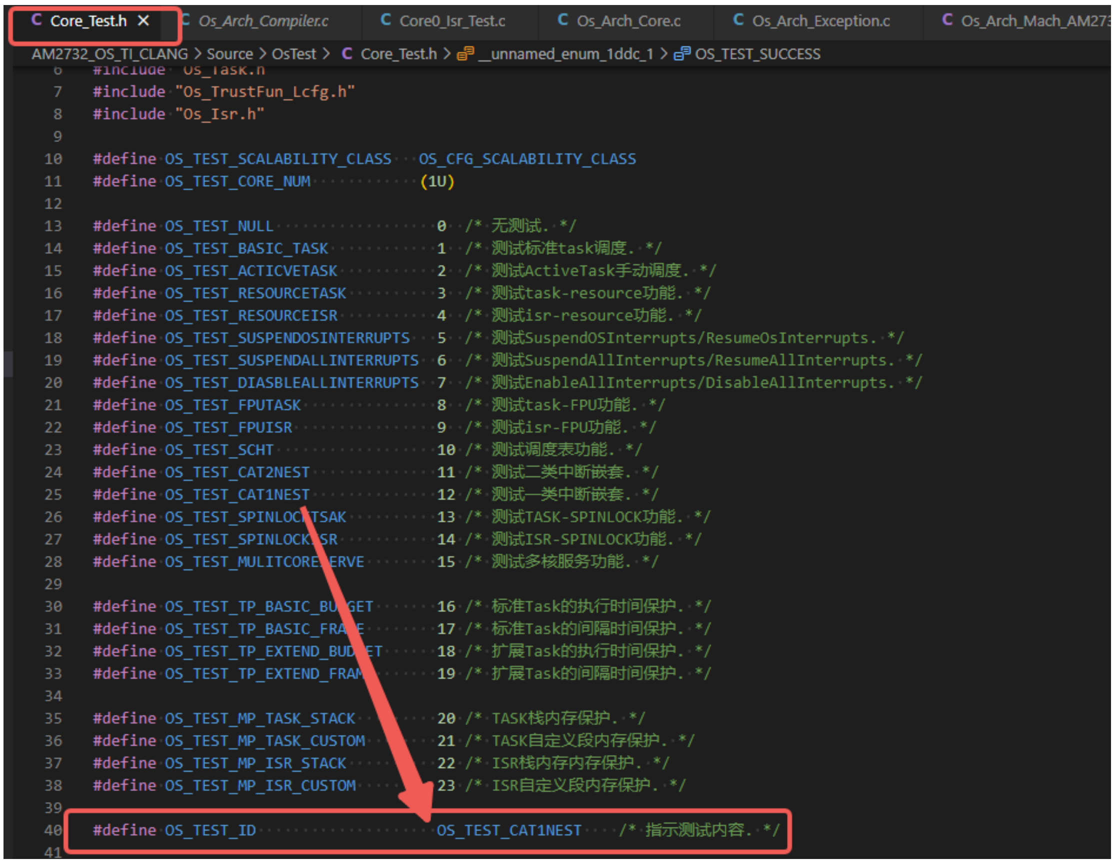
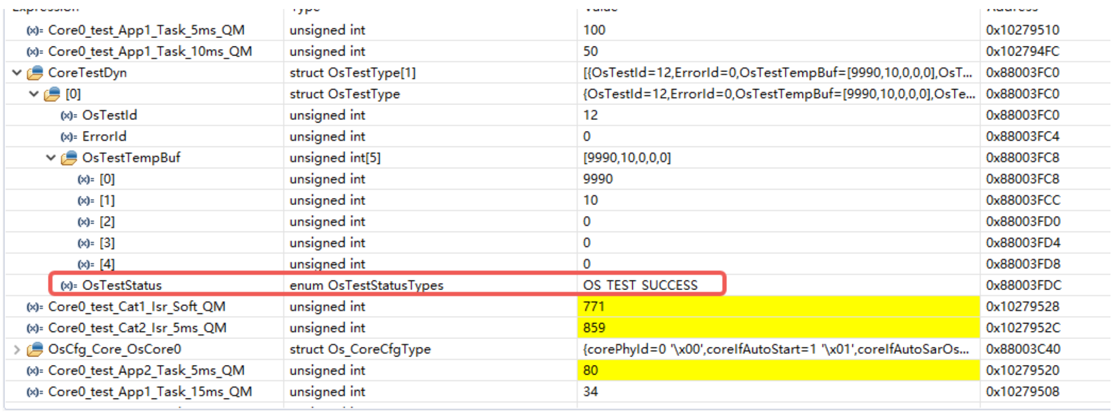
中文版(Chinese Edition):

1. 使用 **Code Composer Studio** 打开并重新编译工程 *Program_SC1_20230329*
2. 修改测试代码，将测试用例修改成 **OS_TEST_CAT1NEST，** 如下图所示:



```
Core_Test.h ×   Os_Arch_Compiler.c   Core0_Isr_Test.c   Os_Arch_Core.c   Os_Arch_Exception.c   Os_Arch_Mach_AM27
AM2732_OS_TI_CLANG > Source > OsTest > Core_Test.h > _unnamed_enum_1ddc_1 > OS_TEST_SUCCESS
#include "Os_Task.h"
#include "Os_TrustFun_Lcfg.h"
#include "Os_Isr.h"

#define OS_TEST_SCALABILITY_CLASS   OS_CFG_SCALABILITY_CLASS
#define OS_TEST_CORE_NUM            (1U)

#define OS_TEST_NULL                   0  /* 无测试. */
#define OS_TEST_BASIC_TASK             1  /* 测试标准task调度. */
#define OS_TEST_ACTICVETASK            2  /* 测试ActiveTask手动调度. */
#define OS_TEST_RESOURCETASK           3  /* 测试task-resource功能. */
#define OS_TEST_RESOURCEISR            4  /* 测试isr-resource功能. */
#define OS_TEST_SUSPENDOSINTERRUPTS    5  /* 测试SuspendOSInterrupts/ResumeOsInterrupts. */
#define OS_TEST_SUSPENDALLINTERRUPTS   6  /* 测试SuspendAllInterrupts/ResumeAllInterrupts. */
#define OS_TEST_DIASBLEALLINTERRUPTS   7  /* 测试EnableAllInterrupts/DisableAllInterrupts. */
#define OS_TEST_FPUTASK                8  /* 测试task-FPU功能. */
#define OS_TEST_FPUISR                 9  /* 测试isr-FPU功能. */
#define OS_TEST_SCHT                   10 /* 测试调度表功能. */
#define OS_TEST_CAT2NEST               11 /* 测试二类中断嵌套. */
#define OS_TEST_CAT1NEST               12 /* 测试一类中断嵌套. */
#define OS_TEST_SPINLOCKTSAK           13 /* 测试TASK-SPINLOCK功能. */
#define OS_TEST_SPINLOCKISR            14 /* 测试ISR-SPINLOCK功能. */
#define OS_TEST_MULITCORESERVE         15 /* 测试多核服务功能. */

#define OS_TEST_TP_BASIC_BUDGET        16 /* 标准Task的执行时间保护. */
#define OS_TEST_TP_BASIC_FRAME         17 /* 标准Task的间隔时间保护. */
#define OS_TEST_TP_EXTEND_BUDGET       18 /* 扩展Task的执行时间保护. */
#define OS_TEST_TP_EXTEND_FRAME        19 /* 扩展Task的间隔时间保护. */

#define OS_TEST_MP_TASK_STACK          20 /* TASK栈内存保护. */
#define OS_TEST_MP_TASK_CUSTOM         21 /* TASK自定义段内存保护. */
#define OS_TEST_MP_ISR_STACK           22 /* ISR栈内存保护. */
#define OS_TEST_MP_ISR_CUSTOM          23 /* ISR自定义段内存保护. */

#define OS_TEST_ID                     OS_TEST_CAT1NEST    /* 指示测试内容. */
```

3. 开始 Debug 工程，运行测试用例观察测试结果



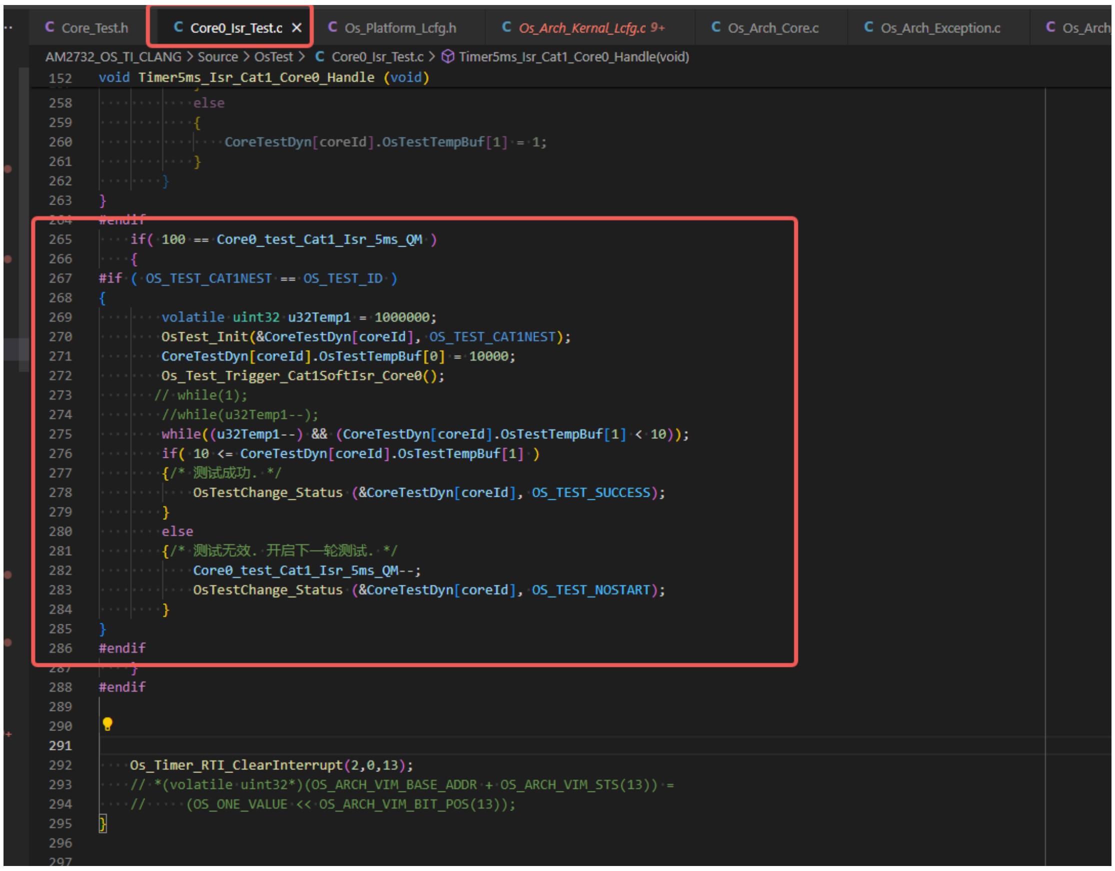
Core0_test_App1_Task_5ms_QM	unsigned int	100	0x10279510
Core0_test_App1_Task_10ms_QM	unsigned int	50	0x102794FC
CoreTestDyn	struct OsTestType[1]	[{OsTestId=12,ErrorId=0,OsTestTempBuf=[9990,10,0,0,0],OsT...	0x88003FC0
[0]	struct OsTestType	{OsTestId=12,ErrorId=0,OsTestTempBuf=[9990,10,0,0,0],OsTe...	0x88003FC0
OsTestId	unsigned int	12	0x88003FC0
ErrorId	unsigned int	0	0x88003FC4
OsTestTempBuf	unsigned int[5]	[9990,10,0,0,0]	0x88003FC8
[0]	unsigned int	9990	0x88003FC8
[1]	unsigned int	10	0x88003FCC
[2]	unsigned int	0	0x88003FD0
[3]	unsigned int	0	0x88003FD4
[4]	unsigned int	0	0x88003FD8
OsTestStatus	enum OsTestStatusTypes	OS_TEST_SUCCESS	0x88003FDC
Core0_test_Cat1_Isr_Soft_QM	unsigned int	771	0x10279528
Core0_test_Cat2_Isr_5ms_QM	unsigned int	859	0x1027952C
OsCfg_Core_OsCore0	struct Os_CoreCfgType	{corePhyId=0 '\x00',coreIfAutoStart=1 '\x01',coreIfAutoSarOs...	0x88003C40
Core0_test_App2_Task_5ms_QM	unsigned int	80	0x10279520
Core0_test_App1_Task_15ms_QM	unsigned int	34	0x10279508

4. 继续运行代码观察变量 Core0_test_App1_Task_5ms_QM 和 Core0_test_App1_Task_10ms_QM 是否会继续计数，如果不能继续计数，则说明系统调度的中断无法正常得到触发。

测试用例说明:

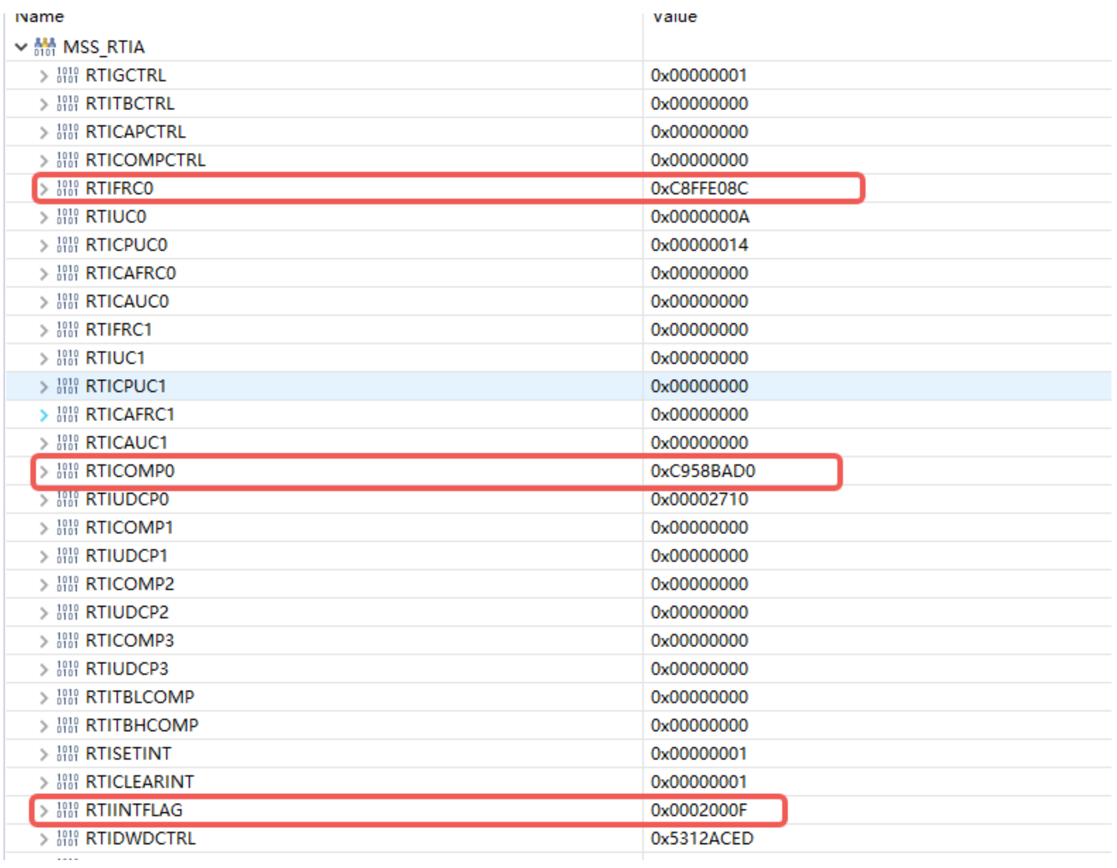
本测试用例为一类中断的嵌套测试，将使用中断服务函数为 Timer5ms_Isr_Cat1_Core0_Handle 的一类中断(IRQ0),该中断的优先级为 2，该中断为周期定时器中断，待该中断触发 100 次后开始主动触发 IRQ2,IRQ2 的中断优先级为 1，IRQ2 也是周期定时器中断，在 IRQ1 中等待 IRQ2 执行 10 次后，认为本条测试用例测试成功，否则测

试失败。部分测试用例截图如下图所示:

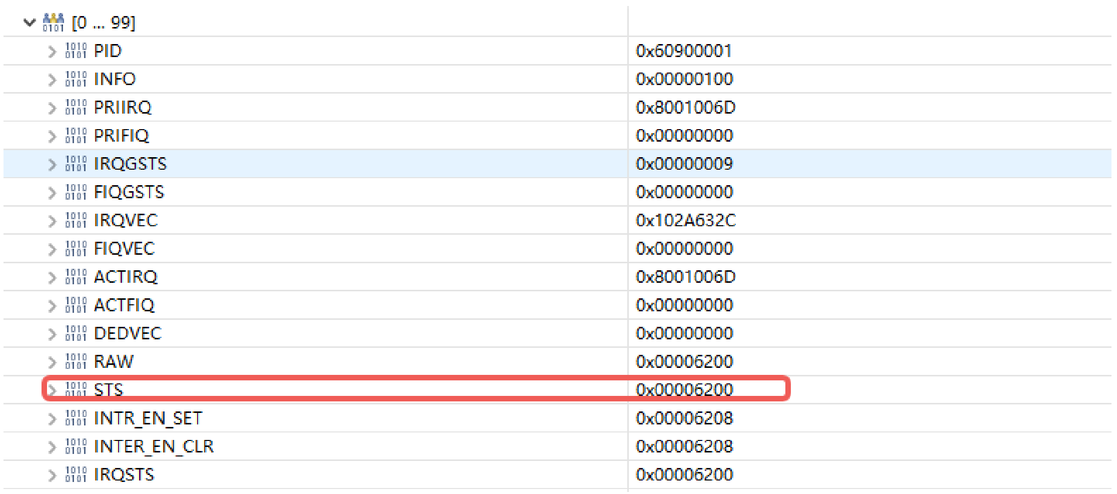


```c
void Timer5ms_Isr_Cat1_Core0_Handle (void)
                else
                {
                    CoreTestDyn[coreId].OsTestTempBuf[1] = 1;
                }
            }
}
#endif
    if( 100 == Core0_test_Cat1_Isr_5ms_QM )
    {
#if ( OS_TEST_CAT1NEST == OS_TEST_ID )
{
        volatile uint32 u32Temp1 = 1000000;
        OsTest_Init(&CoreTestDyn[coreId], OS_TEST_CAT1NEST);
        CoreTestDyn[coreId].OsTestTempBuf[0] = 10000;
        Os_Test_Trigger_Cat1SoftIsr_Core0();
       // while(1);
        //while(u32Temp1--);
        while((u32Temp1--) && (CoreTestDyn[coreId].OsTestTempBuf[1] < 10));
        if( 10 <= CoreTestDyn[coreId].OsTestTempBuf[1] )
        {/* 测试成功. */
            OsTestChange_Status (&CoreTestDyn[coreId], OS_TEST_SUCCESS);
        }
        else
        {/* 测试无效. 开启下一轮测试. */
            Core0_test_Cat1_Isr_5ms_QM--;
            OsTestChange_Status (&CoreTestDyn[coreId], OS_TEST_NOSTART);
        }
}
#endif
    }
#endif



    Os_Timer_RTI_ClearInterrupt(2,0,13);
    // *(volatile uint32*)(OS_ARCH_VIM_BASE_ADDR + OS_ARCH_VIM_STS(13)) =
    //     (OS_ONE_VALUE << OS_ARCH_VIM_BIT_POS(13));
}
```

在执行完该测试用例后，理论上 OS 需要能继续正常运行，包括 Task 都需要能够正常调度，但我们发现，执行该该测试用例后，Task 无法正常调度，现象为上述的两个变量 Core0_test_App1_Task_5ms_QM 和 Core0_test_App1_Task_10ms_QM 无法正常继续增长计数，也就代表着其对应的 Task 无法正常执行。经过调查发现，负责系统调度的时基定时器中断无法再次得到正常触发，该时基定时器使用 RTIA 的 Overflow 模式，其中断 ID 为 3，其优先级为 3。观察 RTIA 对应的寄存器发现，定时器是处于正常运行状态，其对应的外设中断标注位也处于置位状态，但其 VIM 中断控制器对应的中断标志位寄存器却没有指示有中断触发，如下图所示:



Name	Value
MSS_RTIA	
RTIGCTRL	0x00000001
RTITBCTRL	0x00000000
RTICAPCTRL	0x00000000
RTICOMPCTRL	0x00000000
RTIFRC0	0xC8FFE08C
RTIUC0	0x0000000A
RTICPUC0	0x00000014
RTICAFRC0	0x00000000
RTICAUC0	0x00000000
RTIFRC1	0x00000000
RTIUC1	0x00000000
RTICPUC1	0x00000000
RTICAFRC1	0x00000000
RTICAUC1	0x00000000
RTICOMP0	0xC958BAD0
RTIUDCP0	0x00002710
RTICOMP1	0x00000000
RTIUDCP1	0x00000000
RTICOMP2	0x00000000
RTIUDCP2	0x00000000
RTICOMP3	0x00000000
RTIUDCP3	0x00000000
RTITBLCOMP	0x00000000
RTITBHCOMP	0x00000000
RTISETINT	0x00000001
RTICLEARINT	0x00000001
RTIINTFLAG	0x0002000F
RTIDWDCTRL	0x5312ACED

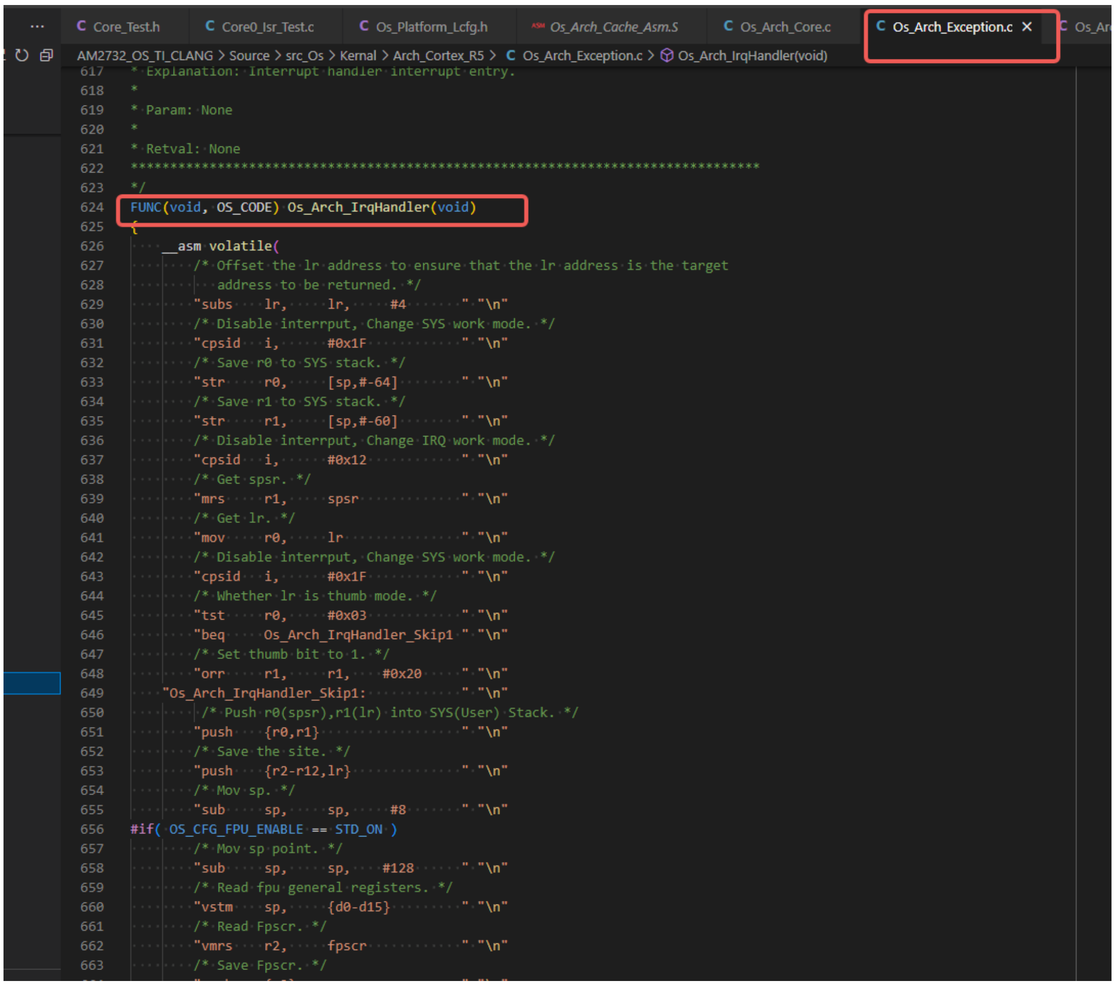


[0 ... 99]	
PID	0x60900001
INFO	0x00000100
PRIIRQ	0x8001006D
PRIFIQ	0x00000000
IRQGSTS	0x00000009
FIQGSTS	0x00000000
IRQVEC	0x102A632C
FIQVEC	0x00000000
ACTIRQ	0x8001006D
ACTFIQ	0x00000000
DEDVEC	0x00000000
RAW	0x00006200
STS	0x00006200
INTR_EN_SET	0x00006208
INTER_EN_CLR	0x00006208
IRQSTS	0x00006200

此外，我们发现，如果将系统调度的时基中断配置成 pluse 触发方式，则没有这种异常现象的产生。

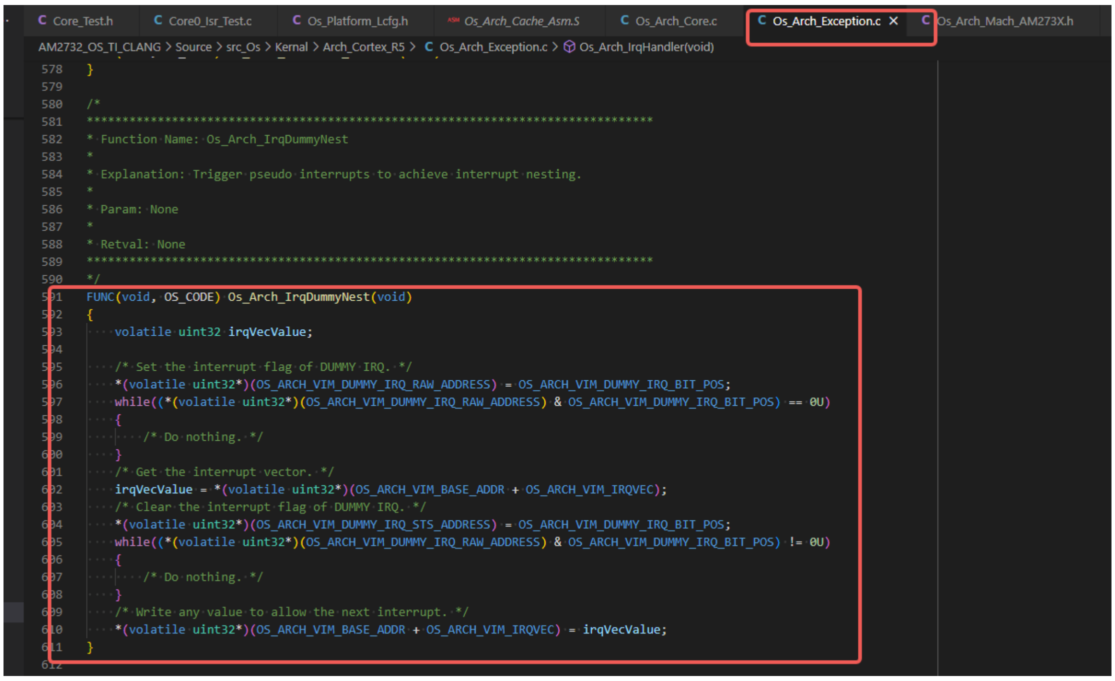
下面我将部分核心代码粘贴在下面，以供参考：

IRQ 中断处理总入口


Core_Test.h
Core0_Isr_Test.c
Os_Platform_Lcfg.h
Os_Arch_Cache_Asm.S
Os_Arch_Core.c
Os_Arch_Exception.c
AM2732_OS_TI_CLANG > Source > src_Os > Kernal > Arch_Cortex_R5 > Os_Arch_Exception.c > Os_Arch_IrqHandler(void)

```c
* Explanation: Interrupt handler interrupt entry.
*
* Param: None
*
* Retval: None
*********************************************************************************
*/
FUNC(void, OS_CODE) Os_Arch_IrqHandler(void)
{
    __asm volatile(
        /* Offset the lr address to ensure that the lr address is the target
           address to be returned. */
        "subs    lr,     lr,     #4       " "\n"
        /* Disable interrput, Change SYS work mode. */
        "cpsid   i,      #0x1F            " "\n"
        /* Save r0 to SYS stack. */
        "str     r0,     [sp,#-64]        " "\n"
        /* Save r1 to SYS stack. */
        "str     r1,     [sp,#-60]        " "\n"
        /* Disable interrput, Change IRQ work mode. */
        "cpsid   i,      #0x12            " "\n"
        /* Get spsr. */
        "mrs     r1,     spsr             " "\n"
        /* Get lr. */
        "mov     r0,     lr               " "\n"
        /* Disable interrput, Change SYS work mode. */
        "cpsid   i,      #0x1F            " "\n"
        /* Whether lr is thumb mode. */
        "tst     r0,     #0x03            " "\n"
        "beq     Os_Arch_IrqHandler_Skip1 " "\n"
        /* Set thumb bit to 1. */
        "orr     r1,     r1,     #0x20    " "\n"
    "Os_Arch_IrqHandler_Skip1:            " "\n"
         /* Push r0(spsr),r1(lr) into SYS(User) Stack. */
        "push    {r0,r1}                  " "\n"
        /* Save the site. */
        "push    {r2-r12,lr}              " "\n"
        /* Mov sp. */
        "sub     sp,     sp,     #8       " "\n"
#if( OS_CFG_FPU_ENABLE == STD_ON )
        /* Mov sp point. */
        "sub     sp,     sp,     #128     " "\n"
        /* Read fpu general registers. */
        "vstm    sp,     {d0-d15}         " "\n"
        /* Read Fpscr. */
        "vmrs    r2,     fpscr            " "\n"
        /* Save Fpscr. */
```

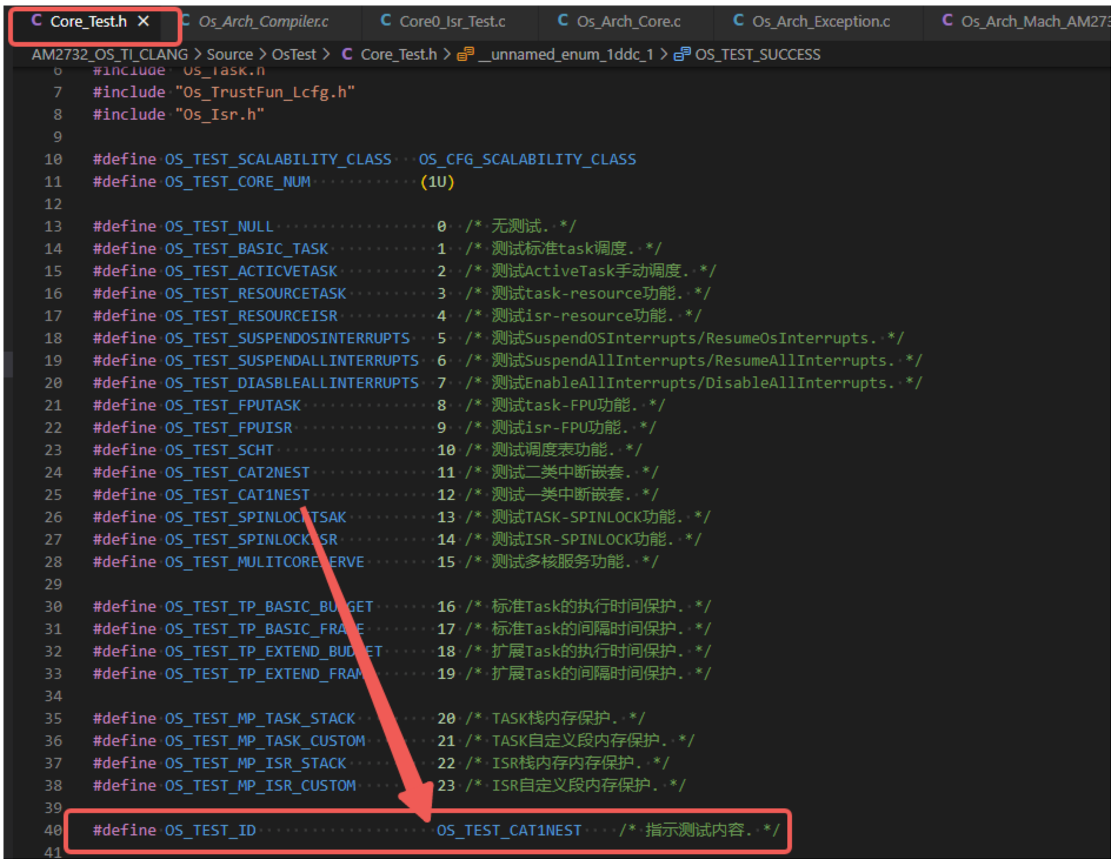
解决中断无法嵌套的软件方案:



```c
}

/*
*********************************************************************************
* Function Name: Os_Arch_IrqDummyNest
*
* Explanation: Trigger pseudo interrupts to achieve interrupt nesting.
*
* Param: None
*
* Retval: None
*********************************************************************************
*/
FUNC(void, OS_CODE) Os_Arch_IrqDummyNest(void)
{
    volatile uint32 irqVecValue;

    /* Set the interrupt flag of DUMMY IRQ. */
    *(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_RAW_ADDRESS) = OS_ARCH_VIM_DUMMY_IRQ_BIT_POS;
    while((*(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_RAW_ADDRESS) & OS_ARCH_VIM_DUMMY_IRQ_BIT_POS) == 0U)
    {
        /* Do nothing. */
    }
    /* Get the interrupt vector. */
    irqVecValue = *(volatile uint32*)(OS_ARCH_VIM_BASE_ADDR + OS_ARCH_VIM_IRQVEC);
    /* Clear the interrupt flag of DUMMY IRQ. */
    *(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_STS_ADDRESS) = OS_ARCH_VIM_DUMMY_IRQ_BIT_POS;
    while((*(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_RAW_ADDRESS) & OS_ARCH_VIM_DUMMY_IRQ_BIT_POS) != 0U)
    {
        /* Do nothing. */
    }
    /* Write any value to allow the next interrupt. */
    *(volatile uint32*)(OS_ARCH_VIM_BASE_ADDR + OS_ARCH_VIM_IRQVEC) = irqVecValue;
}
```

English Edition(英文版):

1. Use **Code Composer Studio** to open and recompile the projec 他 *Program_SC1_20230329*
2. Modify the test code and change the test case to **OS_TEST_CAT1NEST**, as shown in the following figure:

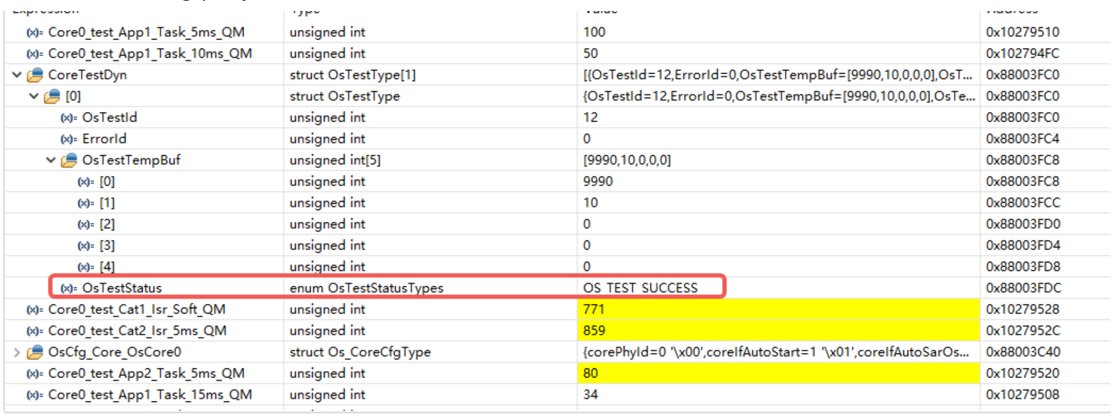


Core_Test.h × | Os_Arch_Compiler.c | Core0_Isr_Test.c | Os_Arch_Core.c | Os_Arch_Exception.c | Os_Arch_Mach_AM27

AM2732_OS_TI_CLANG > Source > OsTest > Core_Test.h > _unnamed_enum_1ddc_1 > OS_TEST_SUCCESS

```c
#include "Os_Task.h"
#include "Os_TrustFun_Lcfg.h"
#include "Os_Isr.h"

#define OS_TEST_SCALABILITY_CLASS   OS_CFG_SCALABILITY_CLASS
#define OS_TEST_CORE_NUM            (1U)

#define OS_TEST_NULL                    0  /* 无测试. */
#define OS_TEST_BASIC_TASK              1  /* 测试标准task调度. */
#define OS_TEST_ACTICVETASK             2  /* 测试ActiveTask手动调度. */
#define OS_TEST_RESOURCETASK            3  /* 测试task-resource功能. */
#define OS_TEST_RESOURCEISR             4  /* 测试isr-resource功能. */
#define OS_TEST_SUSPENDOSINTERRUPTS     5  /* 测试SuspendOSInterrupts/ResumeOsInterrupts. */
#define OS_TEST_SUSPENDALLINTERRUPTS    6  /* 测试SuspendAllInterrupts/ResumeAllInterrupts. */
#define OS_TEST_DIASBLEALLINTERRUPTS    7  /* 测试EnableAllInterrupts/DisableAllInterrupts. */
#define OS_TEST_FPUTASK                 8  /* 测试task-FPU功能. */
#define OS_TEST_FPUISR                  9  /* 测试isr-FPU功能. */
#define OS_TEST_SCHT                    10 /* 测试调度表功能. */
#define OS_TEST_CAT2NEST                11 /* 测试二类中断嵌套. */
#define OS_TEST_CAT1NEST                12 /* 测试一类中断嵌套. */
#define OS_TEST_SPINLOCKTSAK            13 /* 测试TASK-SPINLOCK功能. */
#define OS_TEST_SPINLOCKISR             14 /* 测试ISR-SPINLOCK功能. */
#define OS_TEST_MULITCORESERVE          15 /* 测试多核服务功能. */

#define OS_TEST_TP_BASIC_BUDGET         16 /* 标准Task的执行时间保护. */
#define OS_TEST_TP_BASIC_FRAME          17 /* 标准Task的间隔时间保护. */
#define OS_TEST_TP_EXTEND_BUDGET        18 /* 扩展Task的执行时间保护. */
#define OS_TEST_TP_EXTEND_FRAME         19 /* 扩展Task的间隔时间保护. */

#define OS_TEST_MP_TASK_STACK           20 /* TASK栈内存保护. */
#define OS_TEST_MP_TASK_CUSTOM          21 /* TASK自定义段内存保护. */
#define OS_TEST_MP_ISR_STACK            22 /* ISR栈内存存保护. */
#define OS_TEST_MP_ISR_CUSTOM           23 /* ISR自定义段内存保护. */

#define OS_TEST_ID                      OS_TEST_CAT1NEST    /* 指示测试内容. */
```

3. Start the Debug project, run test cases to observe test results



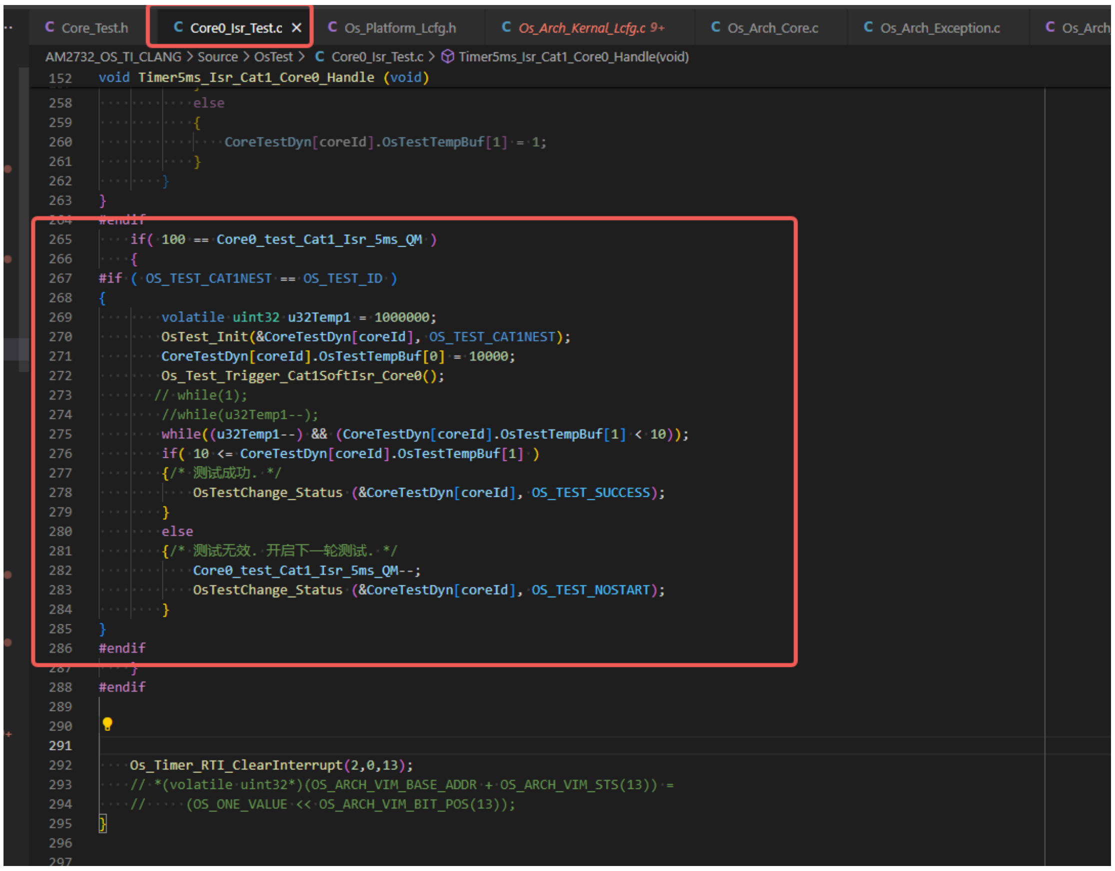
Core0_test_App1_Task_5ms_QM	unsigned int	100	0x10279510
Core0_test_App1_Task_10ms_QM	unsigned int	50	0x102794FC
CoreTestDyn	struct OsTestType[1]	[{OsTestId=12,ErrorId=0,OsTestTempBuf=[9990,10,0,0,0],OsT...	0x88003FC0
[0]	struct OsTestType	{OsTestId=12,ErrorId=0,OsTestTempBuf=[9990,10,0,0,0],OsTe...	0x88003FC0
OsTestId	unsigned int	12	0x88003FC0
ErrorId	unsigned int	0	0x88003FC4
OsTestTempBuf	unsigned int[5]	[9990,10,0,0,0]	0x88003FC8
[0]	unsigned int	9990	0x88003FC8
[1]	unsigned int	10	0x88003FCC
[2]	unsigned int	0	0x88003FD0
[3]	unsigned int	0	0x88003FD4
[4]	unsigned int	0	0x88003FD8
OsTestStatus	enum OsTestStatusTypes	OS_TEST_SUCCESS	0x88003FDC
Core0_test_Cat1_Isr_Soft_QM	unsigned int	771	0x10279528
Core0_test_Cat2_Isr_5ms_QM	unsigned int	859	0x1027952C
OsCfg_Core_OsCore0	struct Os_CoreCfgType	{corePhyId=0 '\x00',coreIfAutoStart=1 '\x01',coreIfAutoSarOs...	0x88003C40
Core0_test_App2_Task_5ms_QM	unsigned int	80	0x10279520
Core0_test_App1_Task_15ms_QM	unsigned int	34	0x10279508

4. Continue running the code to observe whether the variables Core0_test_App1_Task_5ms_QM and Core0_test_App1_Task_10ms_QM will continue counting. If they cannot continue counting, it indicates that the system's scheduled interrupt cannot be triggered properly。

Test Case Specification:

This test case is a nested test of a type of interrupt, which will use a type of interrupt

(IRQ0) with an interrupt service function of Timer5ms_Isr_Cat1_Core0_Handle. The priority of this interrupt is 2, which is a periodic timer interrupt. After the interrupt is triggered 100 times, IRQ2 will be actively triggered. The priority of IRQ2 is 1, and IRQ2 is also a periodic timer interrupt. After waiting for IRQ2 to execute 10 times in IRQ1, the test case is considered successful. Otherwise, the test will fail. Screenshots of some test cases are shown in the following figure:

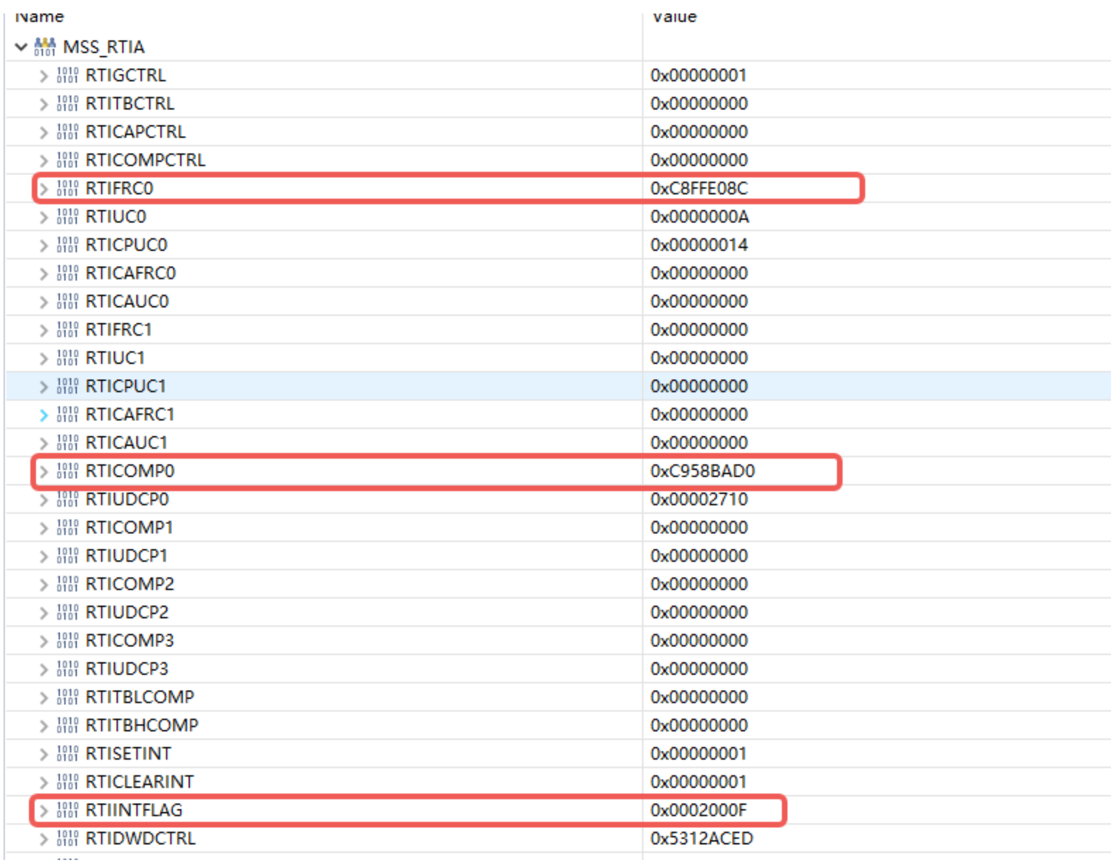


Core_Test.h | Core0_Isr_Test.c | Os_Platform_Lcfg.h | Os_Arch_Kernal_Lcfg.c 9+ | Os_Arch_Core.c | Os_Arch_Exception.c | Os_Arch

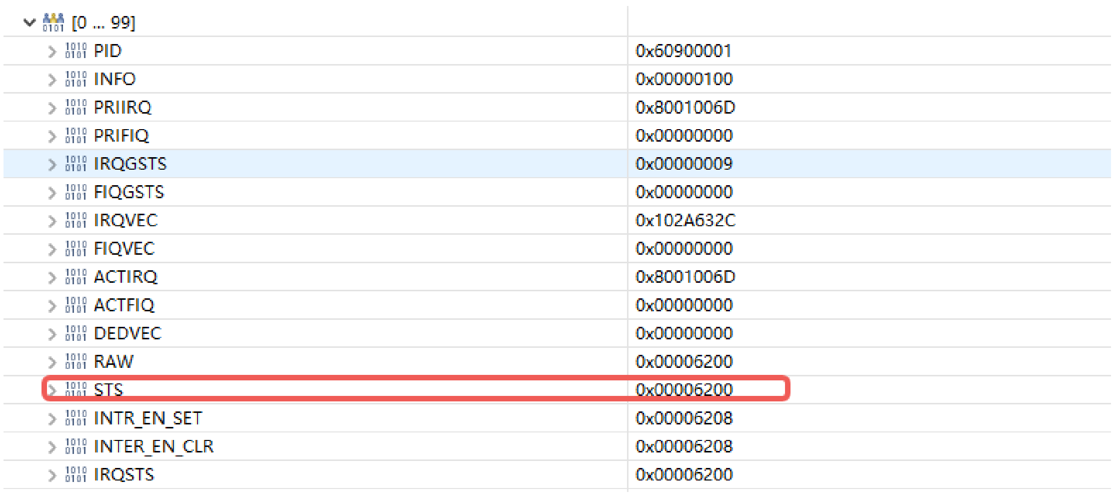
AM2732_OS_TI_CLANG > Source > OsTest > Core0_Isr_Test.c > Timer5ms_Isr_Cat1_Core0_Handle(void)

```c
void Timer5ms_Isr_Cat1_Core0_Handle (void)
                else
                {
                    CoreTestDyn[coreId].OsTestTempBuf[1] = 1;
                }
            }
}
#endif
    if( 100 == Core0_test_Cat1_Isr_5ms_QM )
    {
#if ( OS_TEST_CAT1NEST == OS_TEST_ID )
{
        volatile uint32 u32Temp1 = 1000000;
        OsTest_Init(&CoreTestDyn[coreId], OS_TEST_CAT1NEST);
        CoreTestDyn[coreId].OsTestTempBuf[0] = 10000;
        Os_Test_Trigger_Cat1SoftIsr_Core0();
       // while(1);
        //while(u32Temp1--);
        while((u32Temp1--) && (CoreTestDyn[coreId].OsTestTempBuf[1] < 10));
        if( 10 <= CoreTestDyn[coreId].OsTestTempBuf[1] )
        {/* 测试成功. */
            OsTestChange_Status (&CoreTestDyn[coreId], OS_TEST_SUCCESS);
        }
        else
        {/* 测试无效. 开启下一轮测试. */
            Core0_test_Cat1_Isr_5ms_QM--;
            OsTestChange_Status (&CoreTestDyn[coreId], OS_TEST_NOSTART);
        }
}
#endif
    }
#endif



    Os_Timer_RTI_ClearInterrupt(2,0,13);
    // *(volatile uint32*)(OS_ARCH_VIM_BASE_ADDR + OS_ARCH_VIM_STS(13)) =
    //     (OS_ONE_VALUE << OS_ARCH_VIM_BIT_POS(13));
}
```

After executing this test case, theoretically, the OS needs to be able to continue running normally, including tasks that need to be able to be scheduled properly. However, we found that after executing this test case, tasks cannot be scheduled properly. The phenomenon is that the two variables mentioned above Core0_test_App1_Task_5ms_QM and Core0_test_App1_Task_10ms_QM cannot continue to increase the count normally, which means that their corresponding tasks cannot be executed normally. After investigation, it was found that the timebase timer interrupt responsible for system scheduling cannot be triggered normally again. The timebase timer uses RTIA's Overflow mode, where the interrupt ID is 3 and its priority is 3. Observing the registers corresponding to RTIA, it is found that the timer is in a normal operating state, and its corresponding peripheral interrupt flag bit is also set. However, the interrupt flag bit register corresponding to its VIM interrupt controller does not indicate any interrupt triggering, as shown in the following figure:



Name	Value
MSS_RTIA	
RTIGCTRL	0x00000001
RTITBCTRL	0x00000000
RTICAPCTRL	0x00000000
RTICOMPCTRL	0x00000000
RTIFRC0	0xC8FFE08C
RTIUC0	0x0000000A
RTICPUC0	0x00000014
RTICAFRC0	0x00000000
RTICAUC0	0x00000000
RTIFRC1	0x00000000
RTIUC1	0x00000000
RTICPUC1	0x00000000
RTICAFRC1	0x00000000
RTICAUC1	0x00000000
RTICOMP0	0xC958BAD0
RTIUDCP0	0x00002710
RTICOMP1	0x00000000
RTIUDCP1	0x00000000
RTICOMP2	0x00000000
RTIUDCP2	0x00000000
RTICOMP3	0x00000000
RTIUDCP3	0x00000000
RTITBLCOMP	0x00000000
RTITBHCOMP	0x00000000
RTISETINT	0x00000001
RTICLEARINT	0x00000001
RTIINTFLAG	0x0002000F
RTIDWDCTRL	0x5312ACED

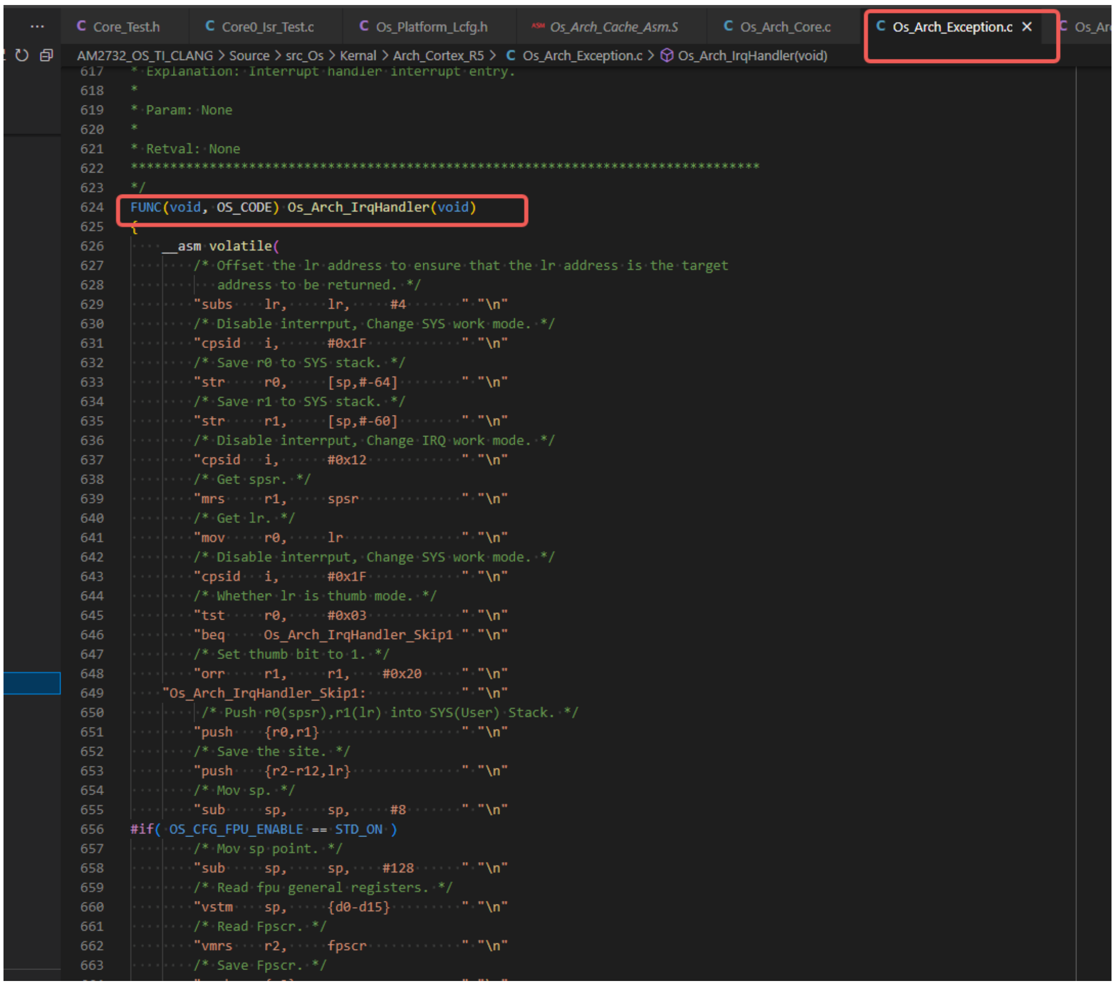


[0 ... 99]	
PID	0x60900001
INFO	0x00000100
PRIIRQ	0x8001006D
PRIFIQ	0x00000000
IRQGSTS	0x00000009
FIQGSTS	0x00000000
IRQVEC	0x102A632C
FIQVEC	0x00000000
ACTIRQ	0x8001006D
ACTFIQ	0x00000000
DEDVEC	0x00000000
RAW	0x00006200
STS	0x00006200
INTR_EN_SET	0x00006208
INTER_EN_CLR	0x00006208
IRQSTS	0x00006200

In addition, we found that if the time base interrupt scheduled by the system is configured as a plug trigger method, there is no occurrence of this abnormal phenomenon。

I will paste some of the core code below for reference:

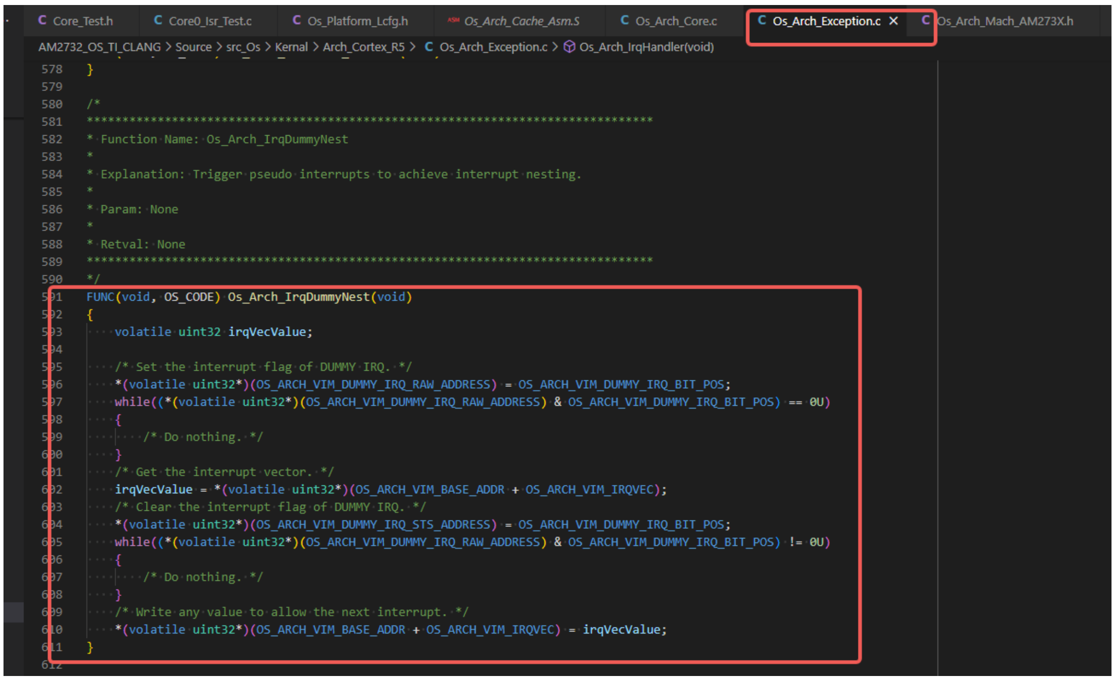
IRQ interrupt processing main entrance



AM2732_OS_TI_CLANG > Source > src_Os > Kernal > Arch_Cortex_R5 > Os_Arch_Exception.c > Os_Arch_IrqHandler(void)

```c
 * Explanation: Interrupt handler interrupt entry.
 *
 * Param: None
 *
 * Retval: None
 ********************************************************************************
 */
FUNC(void, OS_CODE) Os_Arch_IrqHandler(void)
{
    __asm volatile(
        /* Offset the lr address to ensure that the lr address is the target
           address to be returned. */
        "subs    lr,     lr,     #4              " "\n"
        /* Disable interrupt, Change SYS work mode. */
        "cpsid   i,      #0x1F                   " "\n"
        /* Save r0 to SYS stack. */
        "str     r0,     [sp,#-64]               " "\n"
        /* Save r1 to SYS stack. */
        "str     r1,     [sp,#-60]               " "\n"
        /* Disable interrupt, Change IRQ work mode. */
        "cpsid   i,      #0x12                   " "\n"
        /* Get spsr. */
        "mrs     r1,     spsr                    " "\n"
        /* Get lr. */
        "mov     r0,     lr                      " "\n"
        /* Disable interrupt, Change SYS work mode. */
        "cpsid   i,      #0x1F                   " "\n"
        /* Whether lr is thumb mode. */
        "tst     r0,     #0x03                   " "\n"
        "beq     Os_Arch_IrqHandler_Skip1        " "\n"
        /* Set thumb bit to 1. */
        "orr     r1,     r1,     #0x20           " "\n"
    "Os_Arch_IrqHandler_Skip1:                   " "\n"
         /* Push r0(spsr),r1(lr) into SYS(User) Stack. */
        "push    {r0,r1}                         " "\n"
        /* Save the site. */
        "push    {r2-r12,lr}                     " "\n"
        /* Mov sp. */
        "sub     sp,     sp,     #8              " "\n"
#if( OS_CFG_FPU_ENABLE == STD_ON )
        /* Mov sp point. */
        "sub     sp,     sp,     #128            " "\n"
        /* Read fpu general registers. */
        "vstm    sp,     {d0-d15}                " "\n"
        /* Read Fpscr. */
        "vmrs    r2,     fpscr                   " "\n"
        /* Save Fpscr. */
```

A software solution to solve the problem of nested interrupts：



AM2732_OS_TI_CLANG > Source > src_Os > Kernal > Arch_Cortex_R5 > Os_Arch_Exception.c > Os_Arch_IrqHandler(void)

```c
}

/*
********************************************************************************
* Function Name: Os_Arch_IrqDummyNest
*
* Explanation: Trigger pseudo interrupts to achieve interrupt nesting.
*
* Param: None
*
* Retval: None
********************************************************************************
*/
FUNC(void, OS_CODE) Os_Arch_IrqDummyNest(void)
{
    volatile uint32 irqVecValue;

    /* Set the interrupt flag of DUMMY IRQ. */
    *(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_RAW_ADDRESS) = OS_ARCH_VIM_DUMMY_IRQ_BIT_POS;
    while((*(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_RAW_ADDRESS) & OS_ARCH_VIM_DUMMY_IRQ_BIT_POS) == 0U)
    {
        /* Do nothing. */
    }
    /* Get the interrupt vector. */
    irqVecValue = *(volatile uint32*)(OS_ARCH_VIM_BASE_ADDR + OS_ARCH_VIM_IRQVEC);
    /* Clear the interrupt flag of DUMMY IRQ. */
    *(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_STS_ADDRESS) = OS_ARCH_VIM_DUMMY_IRQ_BIT_POS;
    while((*(volatile uint32*)(OS_ARCH_VIM_DUMMY_IRQ_RAW_ADDRESS) & OS_ARCH_VIM_DUMMY_IRQ_BIT_POS) != 0U)
    {
        /* Do nothing. */
    }
    /* Write any value to allow the next interrupt. */
    *(volatile uint32*)(OS_ARCH_VIM_BASE_ADDR + OS_ARCH_VIM_IRQVEC) = irqVecValue;
}
```