

(163条消息) ThreadX内核源码分析(SMP) - 核间通信 (arm)_arm7star的博客-CSDN博客_arm核间通信

 blog.csdn.net/arm7star/article/details/125625592

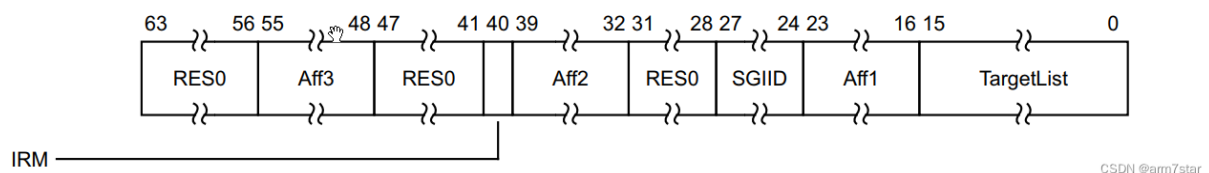
1、ThreadX核间通信介绍

多核情况下，一个核可能改变另外一个核的执行状态，例如一个核挂起另外一个核正在执行的线程，如果没有机制通知另外一个核该线程被挂起的话，那么被挂起的线程就感知不到自己被挂起了；ThreadX内核使用SGI中断，从一个核发送一个中断信号到目的核上去，从而使另外一个核产生中断，另外一个核就会去检查状态变化，例如是否要重新调度(别的核改变了该核需要执行的线程)。

2、发生SGI中断

参考《ARMv8-A_Architecture_Reference_Manual_(Issue_A.a).pdf》"D8.6.28 ICC_SGI1R_EL1, Interrupt Controller Software Generated Interrupt group 1 Register"。

ICC_SGI1R_EL1寄存器如下，24~27位为中断ID(SGIID)，0~15位为目标cpu，每一位代表一个cpu:



ThreadX内核使用0号中断进行核间通信，发送核间中断函数为"`void _tx_thread_smp_core_preempt(UINT core)`", 参数core即为目标cpu id, `_tx_thread_smp_core_preempt`只给一个核发送中断，实现代码如下:

```

1.      .global  _tx_thread_smp_core_preempt

2.      .type    _tx_thread_smp_core_preempt, @function

3. _tx_thread_smp_core_preempt:

4.      DSB ISH

5. #ifdef TX_ARMV8_2

6.      MOV x2, #0x1                                // Build the target list field

7.      LSL x3, x0, #16                              // Build the affinity1 field

8.      ORR x2, x2, x3                                // Combine the fields

9. #else

10.     MOV x2, #0x1                                //

11.     LSL x2, x2, x0                                // Shift by the core ID // x2 = 1 <<
        core

12. #endif

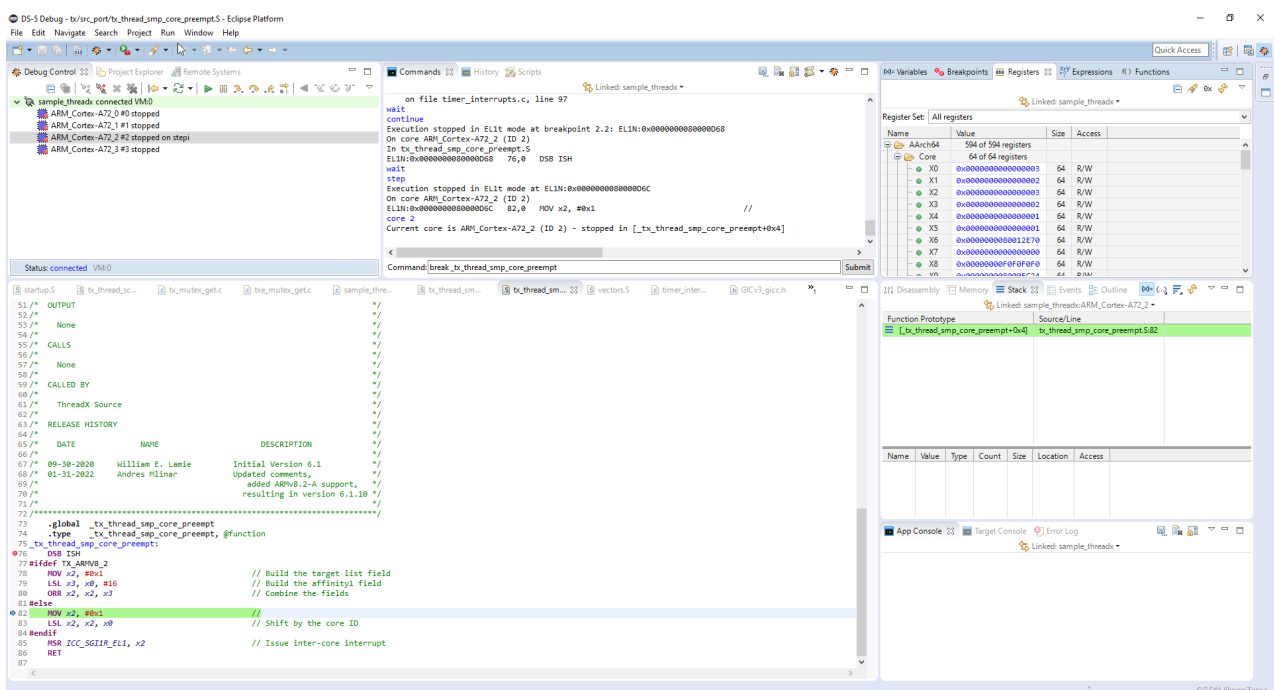
13.     MSR ICC_SGI1R_EL1, x2                        // Issue inter-core interrupt

14.     RET

```

简化成c代码就是“`ICC_SGI1R_EL1 = 1 << core`”，只设置了target list，因为`_tx_thread_smp_core_preempt`发送的是0号中断，因此SGIID默认0即可。

下图所示为从核2发送中断到核3，左上角窗口的"ARM_Cortex-A72_2 #2 stoped on stepi"为当前调试的核，也就是单步执行指令的核，右上角窗口显示的是当前核的寄存器列表，x0也就是`_tx_thread_smp_core_preempt`的参数`o(core)`，core就是3，也就是要从核2发送中断到核3：



3、IRQ中断处理

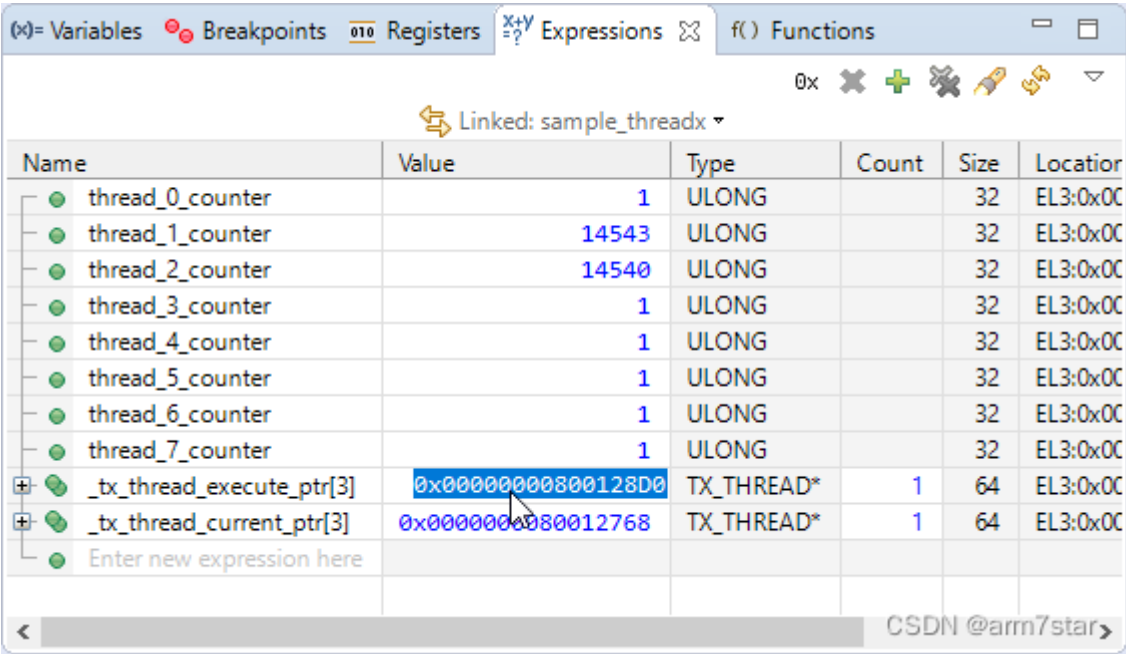
ThreadX的中断处理函数为irqHandler，ThreadX官网提供的样例只处理了核间中断和定时器中断，有这两个中断就可以让ThreadX内核正常运行起来。irqHandler主要就是读取中断号，找到对应的中断处理函数并调用中断处理函数，中断响应，代码如下：

```
1. void irqHandler(void)
2. {
3.     unsigned int ID;
4.     ID = getICC_IAR1(); // readIntAck();
5.     // Check for reserved IDs
6.     if ((1020 <= ID) && (ID <= 1023))
7.     {
8.         //printf("irqHandler() - Reserved INTID %d\n\n", ID);
9.         return;
10.    }
11.    switch(ID)
12.    {
13.        case 34: // 定时器中断(34), 线程时间片以及应用程序定时器需要用到
14.            // Dual-Timer 0 (SP804)
15.            //printf("irqHandler() - External timer interrupt\n\n");
16.            nudge_leds();
17.            clearTimerIrq(); // 清除中断
18.            /* Call ThreadX timer interrupt processing. */
19.            _tx_timer_interrupt(); // 定时器中断处理函数
20.            break;
21.        default: // 其他中断不处理(返回)
22.            // Unexpected ID value
23.            //printf("irqHandler() - Unexpected INTID %d\n\n", ID);
24.            break;
25.    }
26.    // Write the End of Interrupt register to tell the GIC
27.    // we've finished handling the interrupt
28.    setICC_EOIR1(ID); // writeAliasedEOI(ID); // 中断结束
29. }
```



核间中断走的是default分支，没有对应的处理函数，真正起作用的是在中断返回的时候检查是否需要重新调度，发送核间中断的作用基本就是为了让目标cpu重新调度；在整个代码中可以看到，在调用_tx_thread_smp_core_preempt函数之前，基本都是改变了_tx_thread_execute_ptr[i]，i为目标cpu，_tx_thread_smp_core_preempt及发送到id为i的cpu上面去。

如下图所示，_tx_thread_current_ptr[3]为核3上面正在运行的线程(核2挂起的线程，核2只是把该线程的状态等改变了，但是并没有让该线程退出执行)，_tx_thread_execute_ptr[3]为下一个要执行的线程(核2挂起核3上正在执行的线程，选择核3上下一个需要调度的线程，并设置_tx_thread_execute_ptr[3])，在SGI中断退出的时候，检查到_tx_thread_execute_ptr[3]与_tx_thread_current_ptr[3]不相等，就会保存_tx_thread_current_ptr[3]的上下文，也就是将_tx_thread_current_ptr[3]换出cpu，从而真正将线程_tx_thread_current_ptr[3]挂起。



The screenshot shows a debugger's 'Variables' window for a process named 'sample_threadx'. It lists several thread counters and two thread pointers. The thread counters are thread_0_counter through thread_7_counter, all of type 'ULONG' and size 32, located at 'EL3:0x0C'. The thread pointers are _tx_thread_execute_ptr[3] and _tx_thread_current_ptr[3], both of type 'TX_THREAD*' and size 64, located at 'EL3:0x0C'. The values for the pointers are 0x00000000800128D0 and 0x0000000080012768 respectively. The window also has tabs for Breakpoints, Registers, Expressions, and Functions.

Name	Value	Type	Count	Size	Location
thread_0_counter	1	ULONG		32	EL3:0x0C
thread_1_counter	14543	ULONG		32	EL3:0x0C
thread_2_counter	14540	ULONG		32	EL3:0x0C
thread_3_counter	1	ULONG		32	EL3:0x0C
thread_4_counter	1	ULONG		32	EL3:0x0C
thread_5_counter	1	ULONG		32	EL3:0x0C
thread_6_counter	1	ULONG		32	EL3:0x0C
thread_7_counter	1	ULONG		32	EL3:0x0C
_tx_thread_execute_ptr[3]	0x00000000800128D0	TX_THREAD*	1	64	EL3:0x0C
_tx_thread_current_ptr[3]	0x0000000080012768	TX_THREAD*	1	64	EL3:0x0C
Enter new expression here					

4、中断恢复

不管是SGI还是普通中断，中断退出时都会检查是否需要重新调度线程，SGI一般都是为了重新调度，普通中断也可能触发重新调度，所以中断退出都要检查一下是否需要重新调度，主要实现如下图，x8是cpu ID(当前是3)，x0是核上当前执行的线程，x2是当前核下一个需要调度的线程，也就是上一小结的_tx_thread_execute_ptr[3]、_tx_thread_current_ptr[3]，如果不相等就要进行线程切换，需要调度_tx_thread_execute_ptr[3]。

```

LDP    x29, x30, [sp], #16          // Recover x29, x30
ERET                                // Return to point of interrupt

// }
tx_thread_not_nested_restore:

/* Determine if a thread was interrupted and no preemption is required. */
// else if (((_tx_thread_current_ptr) && (_tx_thread_current_ptr == _tx_thread_execute_ptr)
//           || (_tx_thread_preempt_disable))
// {

LDR    x1, =_tx_thread_current_ptr    // Pickup address of current thread ptr
LDR    x0, [x1, x8, LSL #3]           // Pickup actual current thread pointer
CMP    x0, #0                        // Is it NULL?
BEQ    __tx_thread_idle_system_restore // Yes, idle system was interrupted
LDR    x3, =_tx_thread_execute_ptr    // Pickup address of execute thread ptr
LDR    x2, [x3, x8, LSL #3]           // Pickup actual execute thread pointer
CMP    x0, x2                        // Is the same thread highest priority?
BEQ    __tx_thread_no_preempt_restore // Same thread in the execute list,
// no preemption needs to happen
LDR    x3, =_tx_thread_smp_protection // Build address to protection structure
LDR    w3, [x3, #4]                  // Pickup the owning core
CMP    w3, w8                        // Is it this core?
BNE    __tx_thread_preempt_restore   // No, proceed to preempt thread

LDR    x3, =_tx_thread_preempt_disable // Pickup preempt disable address
LDR    w2, [x3, #0]                  // Pickup actual preempt disable flag
CMP    w2, #0                        // Is it set?
BEQ    __tx_thread_preempt_restore   // No, okay to preempt this thread

__tx_thread_no_preempt_restore:

/* Restore interrupted thread or ISR. */

/* Pickup the saved stack pointer. */
// sp = _tx_thread_current_ptr -> tx_thread_stack_ptr;

```