

ETM Trace 调试原理，设置与功能

1. ETM Trace 介绍

ETM 嵌入式跟踪宏单元

ETM 单元用于提供指令跟踪，在使能并开始跟踪后，ETM 会生成指令跟踪包，并通过 ARM CoreSight 调试架构中的跟踪端口接口单元 TPIU 进行输出，跟踪数据通过硬件仿真器传输到 PC 端的调试器软件，调试器软件能够对这些跟踪包进行解析并还原出 MCU 内部的指令执行情况。由于有完整的指令流数据的记录，ETM Trace 可以通过对这些跟踪数据的分析，在分析代码跑飞或异常中断等情形时提供极大的帮助。

1.1 芯片带有 ETM 模块

如上文所述，Trace 功能依赖于 ETM 模块跟踪记录到的 Trace 数据。但是要注意，ETM 硬件模块对于 MCU 而言是一个可选模块，并非所有型号的 MCU 都具有 ETM，所以需要首先查询 MCU 的数据手册确认是否存在 ETM 硬件模块。

1.2 芯片的引脚数量

若查询 MCU 数据手册，确认该型号 MCU 包含 ETM 硬件模块，还需要进一步确认所使用的 MCU 的引脚情况，部分型号 MCU 虽然内部带有 ETM 硬件模块，但是在引脚数量较少的型号中没有包含 Trace 数据的输出引脚，此时同样无法使用 ETM Trace 功能。

1.3 仿真器支持 Trace 功能

Trace 功能会产生大量 Trace 数据，因此需要用到支持 Trace 功能的仿真器例如 I-jet Trace，此类仿真器内部带有专门的存储空间，用于缓存 Trace 数据。不支持 Trace 功能的仿真器如普通版本的 I-jet 仿真器则不支持 ETM Trace 调试功能。



图 1 I-jet 仿真器

1.4 Trace 数据输出引脚和调试接口连接

带有 ETM 硬件模块的 MCU 会具有用于输出 Trace 数据的 GPIO 引脚, 使用 ETM Trace 时, 需要将芯片上的 Trace 数据输出引脚与调试接口连接。常用的 20 Pin JTAG 调试接口下, Trace 数据输出引脚和时钟线对应的连接位置如下图所示:

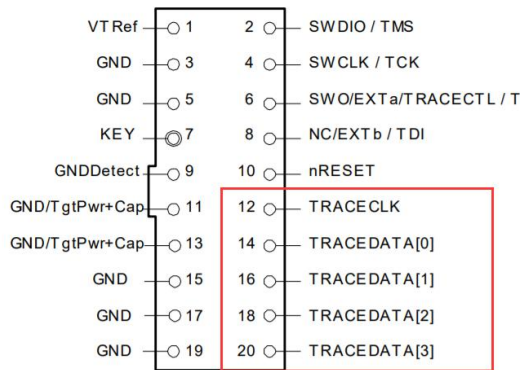


图 2 JTAG 接口

2. ETM Trace 调试功能

借助 ETM Trace, IAR 提供了包括 Trace 指令记录、函数分析、代码覆盖率分析和 Timeline 时间线等强大的调试功能, 下面一一介绍如何使用这些功能。

2.1 Trace 窗口

进入调试界面后, 通过 I-jet->Trace 选项打开如下界面, 将显示通过 Trace 记录到的 CPU 上执行的所有指令, 支持显示包含汇编代码和 C 语句的混合视图, 便于掌握指令的执行情况。在该窗口中会包含一些高亮图标来指示不同的信息, 如函数调用和返回用绿色三角形表示。

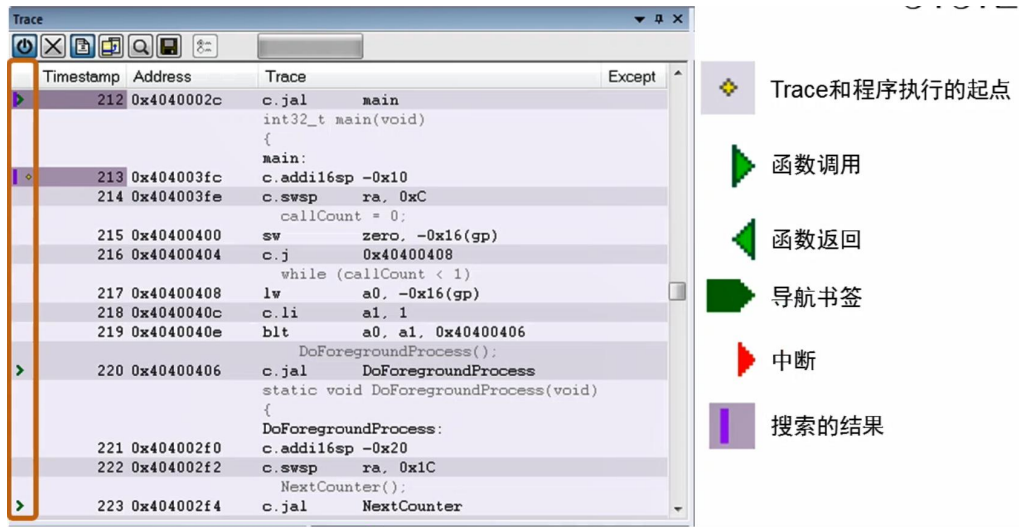


图 3 Trace 窗口

为了便于找到关注的问题位置, 在 Trace 窗口中右键即可打开 Find 搜索窗口, 可以迅速跳转到源码关键位置处。

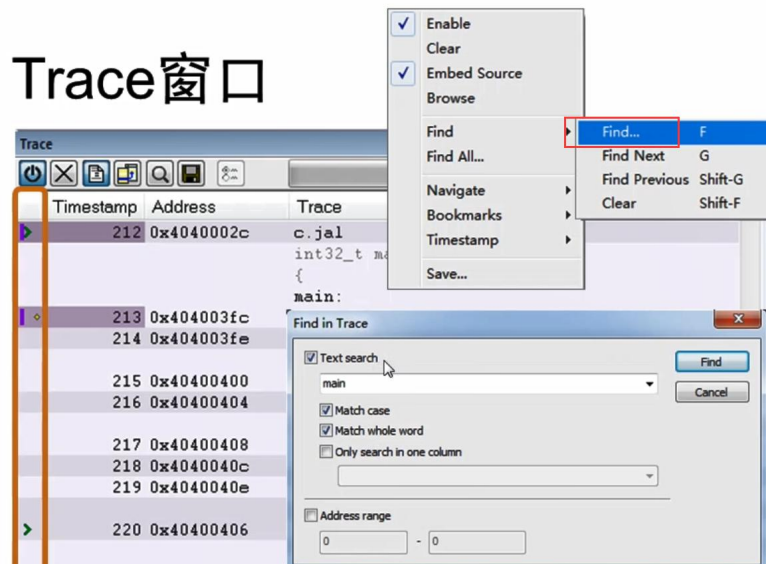


图 4 Trace 的查找功能

2.2 Function Trace 窗口

Function Trace 窗口能够提供函数级别的函数调用和返回关系, 在不想要关注函数内部细节进行问题分析的时候这个窗口非常有用。下图中用红框高亮了 main 函数的调用和返回位置, 蓝色箭头则显示了 DoForegroundProcess 函数的调用和返回位置。

函数级Trace窗口

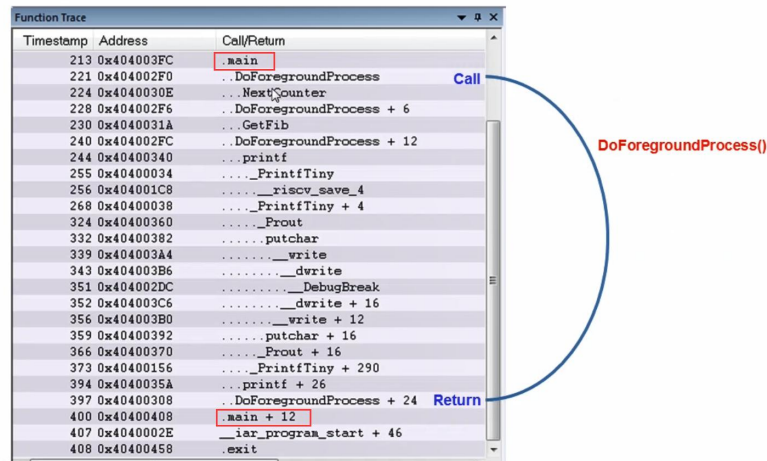


图 5 Function Trace 窗口

2.3 Timeline 时间窗口

除了上述形式的函数分析外，IAR 还提供了 Timeline 视图，能够在时间轴上清晰的展现函数的执行情况，以及各子函数的执行时间，借助 Timeline 视图，可以清晰的了解函数的调用层级关系，以及函数时序和时间占用方面的信息。

Timeline 的时间间隔可以进行调整，可以放大后关注想要了解的函数的执行情况，下图中显示了 printf 函数在执行时子函数的调用层级及执行时间等信息。

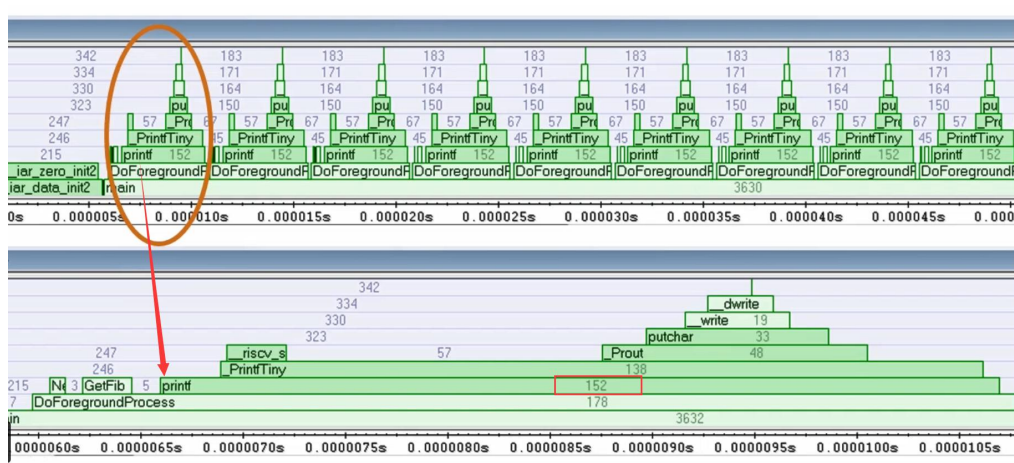
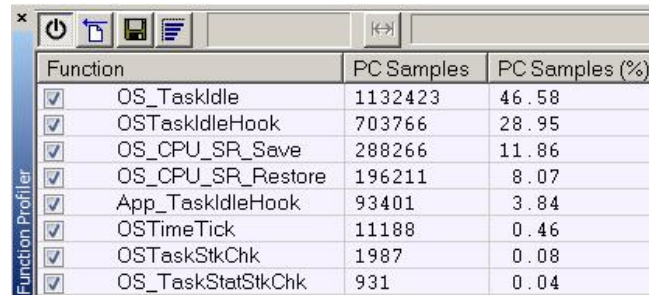


图 6 Timeline 窗口

2.4 函数分析功能

通过 I-jet->Function Profiling 打开函数分析窗口，基于 Trace 记录的数据，可以分析得出执行最多次数的函数，即占用 CPU 运行时间最多的函数，针对性的对这些函数进行优化可以进一步提示系统性能。



Function	PC Samples	PC Samples (%)
OS_TaskIdle	1132423	46.58
OSTaskIdleHook	703766	28.95
OS_CPU_SR_Save	288266	11.86
OS_CPU_SR_Restore	196211	8.07
App_TaskIdleHook	93401	3.84
OSTimeTick	11188	0.46
OSTaskStkChk	1987	0.08
OS_TaskStatStkChk	931	0.04

图 7 Function Profiling 窗口

2.5 函数覆盖率分析

在源码测试过程中，往往需要进行函数覆盖率分析，检查所有的函数是否有被执行到，IAR 函数覆盖率分析能够显示出整个工程、某个 C 文件和某一个函数内部的执行情况，若所有函数和指令都被执行到了，则显示为绿色，未执行到的部分则以红色显示，部分被执行的则是红绿混合。

通过函数覆盖率分析，可以进一步分析哪些函数分支未被执行到，测试团队可以参考函数覆盖率分析结果针对性的修改测试用例，保证源码的质量。

下图中红色显示的函数即未被执行到，点击跳转到对应源码位置进行进一步分析，可以发现此处为一个判断语句且该判断条件一直未成立。

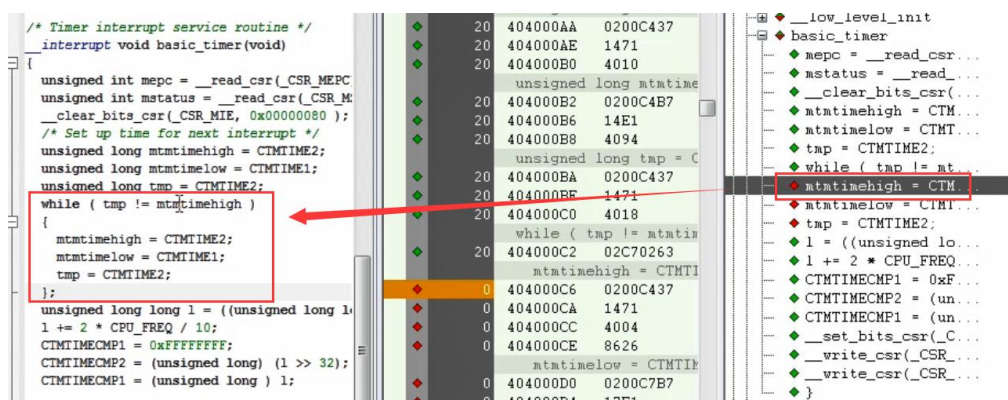


图 8 代码覆盖率窗口