

工训 A413 实验室自制教程 作者：李老色批

工训 A413 自制教程系列

CM3 单片机调试原理浅析

—以 STM32F103 单片机为例

作者：李老色批

目录

- 前言 1
 - (1) 调试功能: 3
 - (2) 跟踪功能: 3
- 一、两大调试接口 4
 - 1. JTAG 接口 4
 - 2. SWD 串行线调试接口 8
- 二、CoreSight 调试接口 9
 - 1. 基于 CoreSight 架构的 CM3 调试系统 9
 - 2. 标准 CoreSight 架构和 CM3 中调试系统异同点 10
- 三、CoreSight 跟踪接口 12
- 四、调试功能的总结 12
- 五、调试模式 13
- 六、调试事件 14
- 七、STM32 调试单元概况 15
- 八、SWV 调试 22
- 附一 JTAG 边界扫描的故事 23
- 附二 代码断点类型 25
- 参考资料 26

前言

“一直以来，单片机的调试一直不是很突出的主题，很多简单些的程序在开发中，甚至都没有调试的概念，而只是把生成的映像直接烧入片子，再根据错误症状来判断问题，然后修改程序重新烧，周而复始，直到问题解决或放弃为止。”

—《Cortex-M3 权威指南》

大部分初学者在学习嵌入式软件时，往往只注重函数 API 的使用和功能的实现，对于调试工具的使用往往不求甚解，殊不知，熟练使用调试工具对嵌入式设备进行故障排查往往是必不可少的技能。

为什么我们需要使用单片机调试？你可能会问这个问题

畅想一下，当代码有好几个分支时，我们想知道当前代码执行的是哪个分支，或代码执行到某一步时，某个变量的值是多少，抑或是我们想知道某个函数被调用了几次、执行时间是多少？

你可能会说：诶，你看我可以用 LED 状态指示灯来判断当前代码的执行情况啊，我还能用这个串口通信输出变量的值，我看那个串口上位机软件接收时间不就能知道代码执行时间了嘛。

没毛病，这个方法的确可以，不过有十分大的限制性：

其一：代码分支复杂情况下无法判断代码执行情况；

其二：无法精确知道代码的执行时间，譬如要求精确到 us 级别；

其三：在功能无法正常运行时，无法进行调试，比如时钟初始化就有问题，后面代码都无法执行。

同时，想象一下如下情况，我们想要同时完成：

(1) 使用 ADC 外设扫描八个通道电压值，采样率为 1KHz，我们想知道每个时间点上八个通道的电压值；

(2) 与此同时，我们还使用了串口外设，我们想知道某个串口在某种情况下某个寄存器的值；

(3) 在某个代码段的某种情况下总是出现 bug，我们想让单片机执行一次代码就暂停一下，同时查看代码中相关变量的值和相关寄存器状态；

(4) 整套程序还使用了 RTOS，我们想知道每个任务的执行时间和执行顺序。

此时，你会发现基于串口通信数据输出的“调试方法”不再适用。

你：不慌，你看，我们可以创建一个定时器中断服务程序，利用串口外设，软件定时解析从 PC 端发送的各种指令，对指令进行解析输出寄存器或者变量的值，对各个任务也可以利用 CPU 主频和多个计数变量完成对任务的执行时间和执行顺序的解析。除了不能一步一步执行程序外，照样可以满足其他功能。



确实，可以，但只限在 MCU 速率较低的情况下，实际上 51 单片机的调试工具 ISD51 在线调试模式下就是这么干的，ISD51 在 8051 系统里增加了一个串口中断函数 (ISD51 中断)。当 ISD51 连接到 uVision 调试器时，8051 输入 ISD51 的中断函数。只要到程序运行一被停止，8051 就只运行 ISD51 中断服务程序。当 uVision2 调试器发出一个“Go”的指令，8051 就脱离 ISD51 中断函数并且运行用户程序。

问题是，该方法会使得程序执行效率大大降低，对于 ISD51 在线模式来说，它的调试会使得程序运行速度将会比原来慢 100 倍，这将使得原本需要高速执行的程序无法正常工作，导致时序出现重大问题。



一般来说，完整的调试系统包括：调试主机、协议转换器和调试目标。

调试主机也就是我们的电脑，调试主机需要配合带有调试功能的 IDE（MDK、IAR、eclipse 等）；协议转换器常用 JLINK、ULINK、STLINK 等，协议转换器一方面用于将调试数据传输到 PC 端的调试器软件，调试器软件能够对这些跟踪数据进行解析并还原出 MCU 内部的指令执行情况，另一方面，协议转换器与目标芯片的调试接口（JTAG、SWD 等）连接，用于数据交互。

调试主机和
调试软件



协议转换器



调试接口

JTAG			
VCC	1	<input type="checkbox"/>	2 VCC (optional)
TRST	3	<input type="checkbox"/>	4 GND
TDI	5	<input type="checkbox"/>	6 GND
TMS	7	<input type="checkbox"/>	8 GND
TCLK	9	<input type="checkbox"/>	10 GND
RTCK	11	<input type="checkbox"/>	12 GND
TDO	13	<input type="checkbox"/>	14 GND
RESET	15	<input type="checkbox"/>	16 GND
N/C	17	<input type="checkbox"/>	18 GND
N/C	19	<input type="checkbox"/>	20 GND

JTAG

SWD			
VCC	1	<input type="checkbox"/>	2 VCC (optional)
N/U	3	<input type="checkbox"/>	4 GND
N/U	5	<input type="checkbox"/>	6 GND
SWDIO	7	<input type="checkbox"/>	8 GND
SWCLK	9	<input type="checkbox"/>	10 GND
N/U	11	<input type="checkbox"/>	12 GND
SWO	13	<input type="checkbox"/>	14 GND
RESET	15	<input type="checkbox"/>	16 GND
N/C	17	<input type="checkbox"/>	18 GND
N/C	19	<input type="checkbox"/>	20 GND

SWD

实际上，一般情况下，基于硬件调试单元的调试系统或组件往往分为以下两种：

调试类别	调试项目	调试特点
侵入式调试	a) 停机以及单步执行程序 b) 硬件断点 c) 断点指令（BKPT） d) 数据观察点，作用于单一地址、一个范围的地址，以及数据的值。 e) 访问寄存器的值（既包括读，也包括写） f) 调试监视器异常 g) 基于 ROM 的调试（闪存地址重载）	调试会打破程序的全速运行
非侵入式调试	a) 在内核运行的时候访问存储器 b) 指令跟踪，需要通过可选的嵌入式跟踪宏单元（ETM） c) 数据跟踪 d) 软件跟踪（通过 ITM（指令跟踪单元）） e) 性能速写（profiling）（通过数据观察点以及跟踪模块）	适用于 RTOS 情况下进行调试

对于 ARM7 和 ARM9 架构的芯片来说，都提供 JTAG 接口，通过它来控制对寄存器和存储器的访问以实现调试功能（参见 1.1.1 节）。

而 CortexM3 内核内含的硬件调试模块是 ARM CoreSight 开发工具集的子集。

ARM CoreSight 开发工具集是 ARM 公司提出的、用于对复杂的 SOC，实现 debug 和 trace 的架构。该架构，包含了多个 CoreSight 组件。众多的 CoreSight 组件，构成了一个 CoreSight 系统。

一个 CoreSight 系统主要功能包括：

(1) 调试功能：

- ①运行处理器的控制，允许启动和停止程序；
- ②单步调试源码和汇编代码；
- ③在处理器运行时设置断点；
- ④即时读取/写入存储器内容和外设寄存器；
- ⑤编程内部和外部 FLASH 存储器。

(2) 跟踪功能：

- ①串行线查看器（SWV）提供程序计数器（PC）采样，数据跟踪，事件跟踪和仪器跟踪信息；
- ②指令（ETM）跟踪直接流式传输到您的 PC，从而实现历史序列的调试，软件性能分析和代码覆盖率分析。

CoreSight 架构定义了调试接口协议、调试总线协议、对调试组件的控制、安全特性、跟踪接口等。在《CoreSight Technology System Design Guide(Ref3)》中，对 CoreSight 有详细的讲述，此外，在 Cortex-M3 技术参考手册中也开出了若干章，专门叙述 CM3 中调试组件的设计。

CM3 调试系统与 ARM7/ARM9 调试系统最大的不同在于：

CM3 对处理器上总线逻辑的控制使用另外的总线接口，即通过所谓的“调试访问端口（DAP）”。DAP 与 AMBA 中的 APB 很相似。在 CM3 中，把 JTAG 或串行线协议都转换成 DAP 总线接口协议，再控制 DAP 来执行调试动作。¹

¹ 关于 AMBA 总线和 AHB 总线协议相关内容可以看如下资料：

Arm AMBA 协议集技术干货汇总：<https://zhuanlan.zhihu.com/p/139328274>

AMBA 总线协议详解：<https://blog.csdn.net/vivid117/article/details/107511746>

一、两大调试接口

1. JTAG 接口

JTAG (IEEE 1149.1 标准) 是联合测试工作组 (Joint Test Action Group) 的简称, 是在名为标准测试访问端口和边界扫描结构的 IEEE 的标准 1149.1 的常用名称。其发明之初, 目的是为了验证设计与测试生产出的印刷电路板功能, **其工作基本原理是在一个点处激励总成的各个导体路径, 而在另一点加以测量。** IEEE 1149.1 标准除描述了支持 JTAG 的设备结构外, 同时定义了描述语言—“边界扫描描述语言 (BSDL)”

目前, JTAG 协议的主要功能包括**边界测试、仿真调试和程序烧写**。常见的 MCU、MPU、DSP、CPLD 和 FPGA 等器件均具有 JTAG-TAP, 即测试访问端口, 该端口用于访问各个板卡, 以进行配置、通信和调试。[关于边界测试内容可以看附一。](#)

具有 JTAG 口的芯片都有如下常用 JTAG 引脚定义:

TCK: 测试时钟输入;

TDI: 测试数据输入, 数据通过 TDI 输入 JTAG 口;

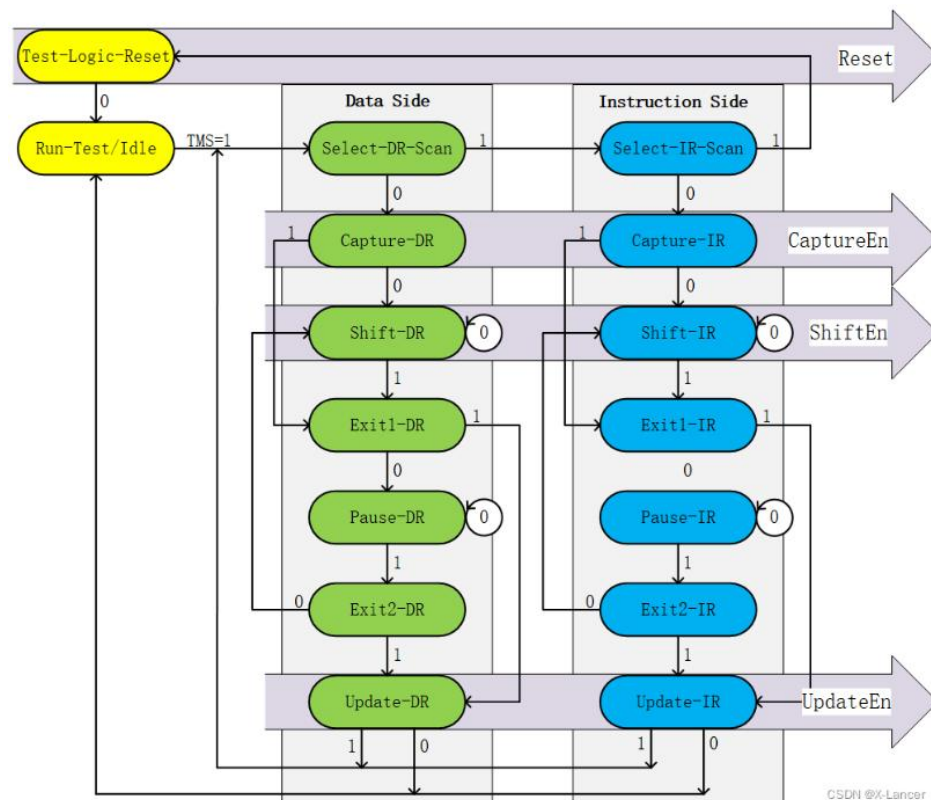
TDO: 测试数据输出, 数据通过 TDO 从 JTAG 口输出;

TMS: 测试模式选择, TMS 用来设置 JTAG 口处于某种特定的测试模式。

可选引脚 TRST: 测试复位, 输入引脚, 低电平有效。

TAP(Test Access Port)是一个通用的端口, 通过 TAP 可以访问芯片提供的所有数据寄存器 (DR)和指令寄存器 (IR)。所有的 JTAG 操作的源头都是由一个具有 16 个状态的同步状态机控制 (即 TAP 控制器) 的。该控制器使用 TCK 为时钟, 使用 TMS 为其输入, 控制器的下一个状态 TMS 信号决定, TMS 信号在 TCK 的上升沿被采样生效, 如果需要复位时使用 TRST。

状态机转换机制如下, 左边绿色的是控制数据寄存器, 右边蓝色的是控制指令寄存器。



TAP 控制器状态	说明
Test-Logic-Reset 测试逻辑复位状态	<p>处于这种状态下，测试逻辑被禁止以允许芯片正常操作，读 IDCODE 寄存器将禁止测试逻辑。</p> <p>无论 TAP 控制器处于何种状态，只要将 TMS 信号在 5 个连续的 TCK 信号的上升沿保持高电平，TAP 就将进入测试逻辑复位状态，如果 TMS 信号一直为高电平，那么 TAP 将保持在测试逻辑复位状态，另外 TRST 信号也可以强迫 TAP 进入测试逻辑复位状态。</p> <p>处于测试逻辑复位状态的 TAP，如果下一个 TCK 的上升沿时 TMS 信号处于低电平，那么 TAP 将被切换到运行测试空闲状态。</p>
Run-Test-Idle 运行测试空闲状态	<p>Run-Test-Idle 是 TAP 控制器扫描操作空闲状态，如果 TMS 信号一直处于低电平，那么 TAP 将保持在 TRun-Test-Idle 状态。当 TMS 信号在 TCK 上升沿处于高电平，TAP 控制器将进入 Select-DR-Scan 状态。</p>
Select-DR-Scan 选择数据寄存器扫描状态	<p>Select-DR-Scan 是 TAP 控制器的一个临时状态，边界扫描寄存器 BSR 保持它们先前的状态。</p> <p>当 TMS 信号在下一个 TCK 上升沿处于低电平，TAP 控制器进入 Capture-DR 状态，一个边界扫描寄存器的扫描操作同时被初始化。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 控制器将进入 Select-IR-Scan 状态。</p>
Capture-DR 捕获数据寄存器状态	<p>如果 TAP 控制器处于 Capture-DR 状态，且当前指令是 SAMPLE 或 PRELOAD 指令，那么边界扫描寄存器 BSR 在 TCK 信号的上升沿捕获输入管脚的数据。如果此时不是 SAMPLE/PRELOAD 指令，那么 BSR 保持它们先前的值，另外 BSR 的值被放入连接在 TDI 和 TDO 管脚之间的移位寄存器中。</p> <p>处于 Capture-DR 状态时，指令不会被改变。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Exit1-DR 状态。如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 进入 Shift-DR 状态。</p>
Shift-DR 移位数据寄存器状态	<p>在 Shift-DR 状态下，在每个 TCK 的上升沿，TDI-移位寄存器-TDO 串行通道向右移一位，TDI 的数据移入移位寄存器，移位寄存器最靠近 TDO 的位移到 TDO 管脚上。</p> <p>处于 Shift-DR 状态时，指令不会被改变。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Exit1-DR 状态。如果 TMS 信号处于低电平，则 TAP 一直进行移位操作。</p>
Exit1-DR 退出数据寄存器状态 1	<p>Exit1-DR 是 TAP 控制器的一个临时状态，如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Update-DR 状态；如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 进入 Pause-DR 状态。</p> <p>处于 Exit1-DR 状态时，指令不会被改变。</p>

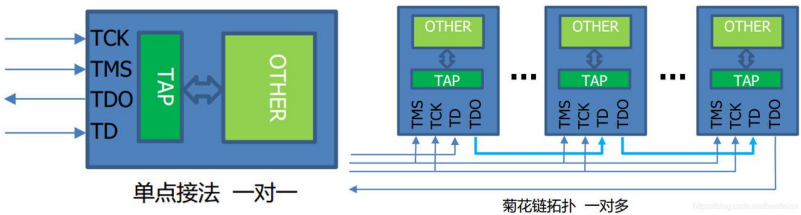
<p>Pause-DR 暂停数据寄存器状态</p>	<p>Pause-DR 状态允许 TAP 控制器暂时停止 TDI-移位寄存器-TDO 串行通道的移位操作。处于 Pause-DR 状态时，指令不会被改变。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Exit2-DR 状态；如果 TMS 信号处于低电平，则 TAP 一直保持暂停状态。</p>
<p>Exit2-DR 退出数据寄存器状态 2</p>	<p>Exit2-DR 也是 TAP 控制器的临时状态，如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Update-DR 状态，结束扫描操作；如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 重新进入 Shift-DR 状态。</p> <p>处于 Exit2-D 状态时，指令不会被改变。</p>
<p>Update-DR 更新数据寄存器状态</p>	<p>在正常情况下，边界扫描寄存器 BSR 的值是被锁存在并行输出管脚中，以免在 EXTEST 或 SAMPLE/PRELOAD 命令下执行移位操作时改变 BSR 的值。当处于 Update-DR 状态时选择的是 BSR 寄存器，那么移位寄存器中的值将在 TCK 的下降沿被锁存到 BSR 寄存器的并行输出管脚中去。</p> <p>处于 Update-DR 状态时，指令不会被改变。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Select-DR-Scan 状态；如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 进入 Run-Test-Idle 状态。</p>
<p>Select-IR-Scan 选择指令寄存器扫描状态</p>	<p>Select-IR-Scan 是 TAP 控制器的一个临时状态。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于低电平，TAP 控制器进入 Capture-IR 状态，一个对指令寄存器的扫描操作同时被初始化。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 控制器将进入 Test-Logic-Reset 状态。</p> <p>处于 Select-IR-Scan 状态时，指令不会被改变。</p>
<p>Capture-IR 捕获指令寄存器状态</p>	<p>处于 Capture-IR 状态时，指令寄存器中的值被固定设置成 0b0000001，并将它放入连接在 TDI 与 TDO 之间的移位寄存器中。</p> <p>处于 Capture-DR 状态时，指令不会被改变。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Exit1-IR 状态；如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 进入 Shift-IR 状态。</p>
<p>Shift-IR 移位指令寄存器状态</p>	<p>在 Shift-IR 状态下，在每个 TCK 的上升沿，TDI-移位寄存器-TDO 串行通道向右移一位，JTAG 指令从 TDI 管脚上被逐位移入移位寄存器，而移位寄存器中的 0b0000001 则被逐位从 TDO 管脚移出。</p> <p>处于 Shift-IR 状态时，指令不会被改变。如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Exit1-IR 状态；如果 TMS 信号处于低电平，则 TAP 一直进行移位操作。</p>

Exit1-IR 退出指令寄存器状态 1	<p>Exit1-IR 是 TAP 控制器的一个临时状态，如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Update-IR 状态；如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 进入 Pause-IR 状态。</p> <p>处于 Exit1-IR 状态时，指令不会被改变。</p>
Pause-IR 暂停指令寄存器状态	<p>Pause-IR 状态允许 TAP 控制器暂时停止 TDI-移位寄存器-TDO 串行通道的移位操作。</p> <p>处于 Pause-IR 状态时，指令不会被改变。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Exit2-IR 状态；如果 TMS 信号处于低电平，则 TAP 一直处于暂停状态。</p>
Exit2-IR 退 出指令寄存器状态 2	<p>Exit2-IR 也是 TAP 控制器的临时状态，如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Update-IR 状态，结束扫描操作；如果 TMS 信号在下一个 TCK 上升沿处于低电平，则 TAP 重新进入 Shift-IR 状态。</p> <p>处于 Exit2-D 状态时，指令不会被改变。</p>
Update-IR 更新指令寄存器状态	<p>处于 Update-IR 状态时，移位寄存器中的值将在 TCK 的下降沿被锁存到指令寄存器中，一旦锁存成功，新的指令将成为当前的指令。</p> <p>如果 TMS 信号在下一个 TCK 上升沿处于高电平，TAP 进入 Select-DR-Scan 状态；如果 TMS 信号在下一个 TCK 上升沿处于电平，则 TAP 进入 Run-Test-Idle 状态。</p>

PC 端通过调试器连接 TAP 接口实现对目标芯片相关寄存器的读取过程简要概况如下：

系统上电后，TAP 控制器首先进入 Test-LogicReset 状态，然后依次进入 Run-Test / Idle、Select-DR- Scan、Select-IR-Scan、Capture-IR、Shift-IR、Exit1-IR、Update-IR 状态，最后回到 Run-Test / Idle 状态。在此过程中，状态的转移都是通过 TCK 信号进行驱动(上升沿)，通过 TMS 信号对 TAP 的状态进行选择转换的。其中，在 Capture-IR 状态下，一个特定的逻辑序列被加载到指令寄存器中；在 Shift-IR 状态下，可以将一条特定的指令送到指令寄存器中；在 Update-IR 状态下，刚才输入到指令寄存器中的指令将用来更新指令寄存器。最后，系统又回到 Run-Test / Idle 状态，指令生效，完成对指令寄存器的访问。当系统又返回到 Run-Test / Idle 状态后，根据前面指令寄存器的内容选定所需要的数据寄存器，开始执行对数据寄存器的工作。其基本原理与指令寄存器的访问完全相同，依次为 Select-DR-Scan、Capture-DR、Shift-D、Exit1-DR、Update-DR，最后回到 Run-Test / Idle 状态。通过 TDI 和 TDO，就可以将新的数据加载到数据寄存器中。经过一个周期后，就可以捕获数据寄存器中的数据，完成对与数据寄存器的每个寄存器单元相连的芯片引脚的数据更新，也完成了对数据寄存器的访问。

这里，特别注意，JTAG 接口不仅支持一对一连接，也支持多个器件通过 JTAG 接口串联在一起，形成一个 JTAG 链，能实现对各个器件分别测试（主要针对多核芯片），其拓扑结构图如下所示。



2. SWD 串行线调试接口

由于当前电路板集成密度不断增高、IC 芯片可用引脚不断减少，与此同时传统的 JTAG 调试接口必须使用 VCC、GND 电源信号，以及 TMS、TCK、TDI、TDO 四根调试信号，另外 TRST、RESET 复位信号和 RTCK（同步时钟）信号也在可选项里，基于以上因素，ARM 创建了一个名为 SWD(串行线调试)的替代调试接口。相对于 JTAG 接口，SWD 使用了更少的信号。它只使用两个信号(SWDCLK 和 SWDIO)，这个接口及其相关协议现在几乎可以在所有的 Cortex-[A,R,M]处理器中使用。

SWD 协议分为两版本，当前所有大多数调试设备基于 SWDV2。

SWD 协议	特点
SWD 协议 v1	<p>点对点架构(point-to-point)，支持单一主机(Host)和单一设备(Device)的连接.主机可以通过建立其他的连接以连接到其他设备.其存在着一些缺点：</p> <p>多个设备在物理连接上存在一定不便</p> <p>增加了设备所需的外部引脚数，同时对芯片内部的 Die 也有要求</p> <p>调试多种平台对于单种控制器的集成要求较高 总的来讲就是同时调试不同平台的设备比较麻烦。</p>
SWD 协议 v2	<p>SWD 协议 2 旨在解决多个串行调试设备之间的连接问题，但同时引入了限制最大连接速度的 trade off。</p> <p>SWD 协议 2 采用多点架构(multi-drop)，具有以下特性：</p> <p>主机仅需两条线即可与多个设备同时通信</p> <p>允许的有效连接数不设限(仅受本身电气特性限制)</p> <p>在最大程度上支持向后兼容(点对点架构本来就是多点结构的一部分)</p> <p>在没有选中设备时，允许设备完全关闭电源防止多个设备对总线的竞争操作，支持总线被主动驱动为高电平或低电平，进而保持较高的最大时钟速度</p> <p>在多点连接中，允许接入不支持 SWD 协议的设备</p>

SWD 协议使用以下两种信号线完成调试器（主机）和设备端的通信：

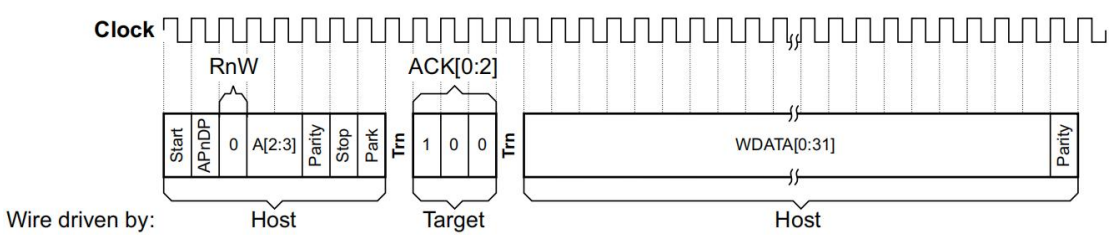
（1）SWDCLK：主机发出的时钟信号，由于处理器时钟和 SWD 时钟之间没有关系，所以频率的选择取决于主机接口。

（2）SWDIO：这是把数据从 DP 传送到 DP 的双向信号，数据由主机在上升沿设置，由 DP 在 SWDCLK 信号下降沿采样。

每个 SWD 交换过程分三个阶段：

- 1.请求阶段：从主机端口发送 8 位；
- 2.ACK 阶段：从目标端口发送 3 位；
- 3.数据阶段：发向主机端口或从主机端口发送最多 32 位，带有奇偶校验位；

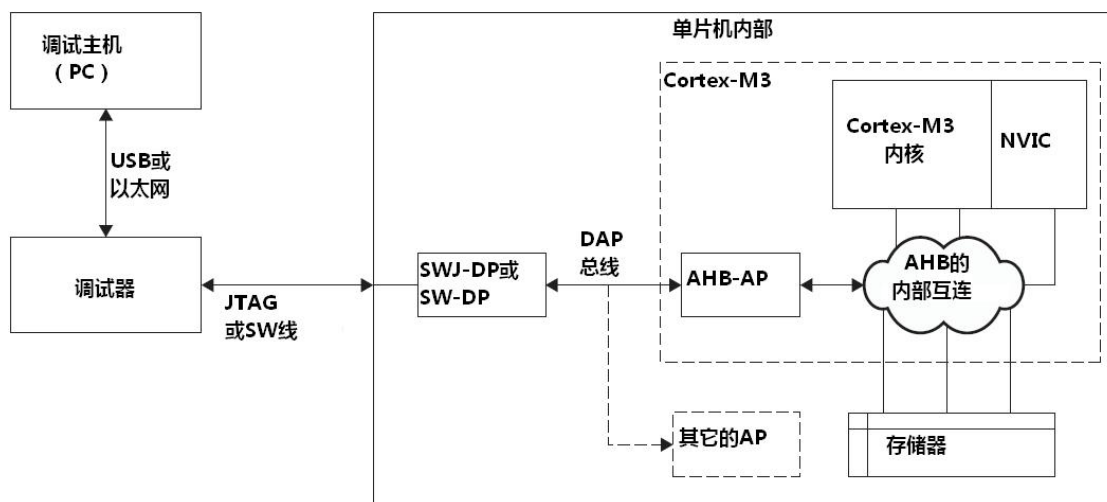
以下为一次成功写操作的时序转换图：



进一步了解 SWD 协议可以参考 [Arm® Debug Interface Architecture Specification ADIv6.0](#)

二、CoreSight 调试接口

1. 基于 CoreSight 架构的 CM3 调试系统



在上文中已经提到，**CM3 架构将调试接口（JTAG 和 SWD）和调试硬件分开，二者通过调试访问端口(DAP)和调试总线相互连接**。这样做的好处在于：芯片中实际使用的调试接口类型变得透明化，从而在执行调试任务时，可以不用理会到底使用了什么调试接口，从而全力以赴地只做自己的事。

在 CM3 处理器内核中，实际的调试功能由 NVIC 和若干调试组件来协作完成。调试组件包括 FPB, DWT, ITM 等。NVIC 中有一些寄存器，用于控制内核的调试动作，如停机、单步；其它的一些功能块则控制观察点、断点，以及调试消息的输出。

就目前来看，CM3 支持两种调试主机接口（debug host interface）：JTAG 接口和串行线调试接口。ARM 公司还提供了若干种调试主机接口模块（称为“调试接口”（DP））。DP 可以实现为 JTAG 调试端口（JTAG-DP）、串行线调试端口（SW-DP）或组合使用串行线/JTAG 调试端口（SWJ-DP）。**DP 充当处理器与调试器的中介：它的一端连接到调试器上，另一端则连接到 CM3 的 DAP 接口上。**

AP、DAP、DP 的概念似乎很容易让人产生困惑。**这里，DP 接口指的是调试接口（Debug Port），AP（Access Port）接口指的是访问接口，DP 接口和 AP 接口合起来称为 DAP 接口。**他们之间的关系是怎么样呢？这里就要说到调试系统的多级互联了。

如上图所示，从外部调试器到 CM3 调试接口的连接，需要通过 DP 接口和 AP 接口，其中：**DP 为调试器提供了通用接口，以访问 AP 中保存的信息；AP 使用特定资源的传输机制来访问待调试系统中的调试信息，并使用规定的访问机制将信息传递给 DP。**有时又将 DAP 接口称为 ADI 接口。

简单的说，DP 和 APs 之间的连接将执行以下的任务：

- （1）根据调试器提供的地址信息，选择相应的调试资源；**
- （2）在 AP 与 DP 之间传输数据。**

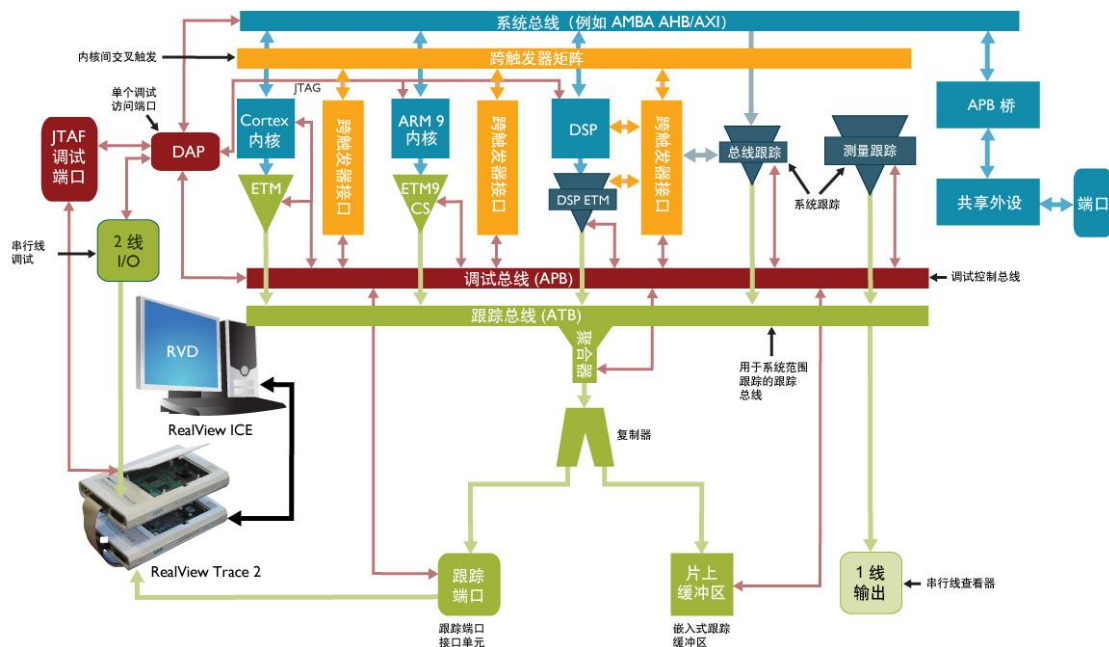
二者数据交互具体流程可以概况如下：

第一步，是通过 DP 接口模块（通常是 SWJ-DP 或 SW-DP），先把外部信号转换成一个通用的 32 位调试总线信号（图表中的 DAP 总线）。SWJ-DP 支持 SW 与 JTAG 两种协议，而 SW-DP 则只支持 SW。另外，在 CoreSight 产品中还可以使用一种 JTAG-DP，它只支持 JTAG 协议。DAP 总线上的地址是 32 位的，其中高 8 位用于选择访问哪一个设备，由此可见最多可以在 DAP 总线上挂 256 个设备。在 CM3 处理器的内部，只用掉了一个设备的地址，还剩下的 255 个都可以用于连接访问端口（AP）到 DAP 总线上。

在把数据从 DAP 接口传递给 CM3 处理器后，下一步就连接到了一个称为“AHB-AP”的 AP 设备上，它相当于一个总线桥²，用于把 DAP 总线的命令转换为 AHB 总线上的数据传送，再插入到 CM3 内部的总线网络中。这么一来，CM3 的整个存储器映射都可以访问了，连 NVIC 中的调试控制寄存器组也包括在内。在 CoreSight 系列产品中，AP 设备可以有好几种类型，包括 APB-AP 和 JTAG-AP。APB-AP 顾名思义，是用于产生 APB 总线数据传送动作的，而 JTAG-AP 则用于控制传统的，基于 JTAG 的测试接口，例如 ARM7 上的调试接口。

2. 标准 CoreSight 架构和 CM3 中调试系统异同点

在标准的 CoreSight 架构中，为调试总线另开了一个地址空间。可以看到，DAP 接收外部端口的 JTAG/SWD 数据，然后转化成对 DAP 内部的 AP 的访问，然后 AP 再转化为 AHB 的总线访问（即 AHB-AP）。AHB-AP 通过 AHB 互联矩阵连接到 Debug APB 互联上，在 Debug APB 互联上连接了有 CTI、ETM、HTM、ITM、ETB、TPIU 等 CoreSight 组件，外部调试器可通过调试接口对这些 CoreSight 组件进行访问；同时 AHB-AP 通过 AHB 互联矩阵连接到系统总线上，系统总线与 APB 总线相连接，以此外部调试器可通过调试接口对外设进行访问。

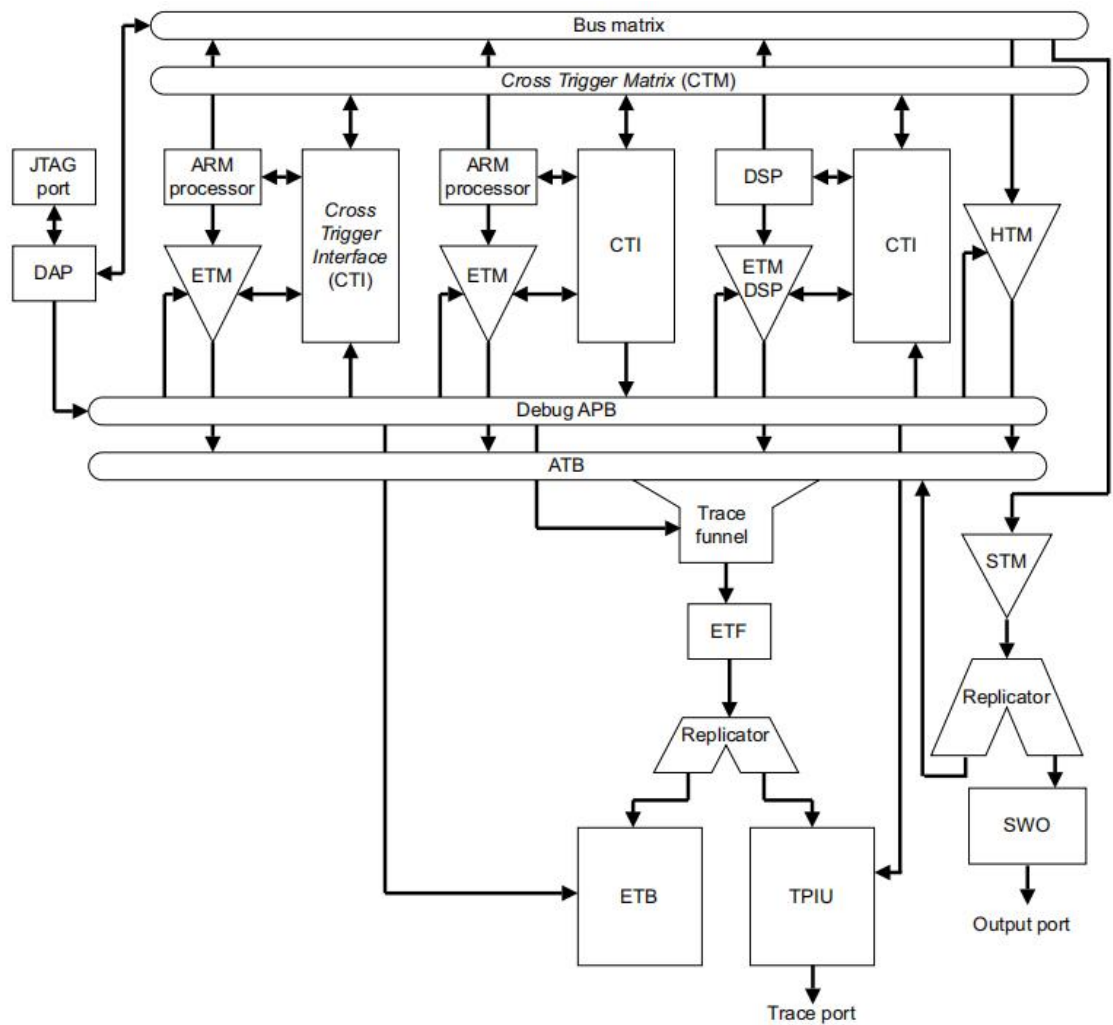


² 总线之间的所谓桥，简单来说就是一个总线转换器，它实现各类微处理器总线到 PCI 总线、各类标准总线到 PCI 总线的连接，并允许它们之间相互通信。

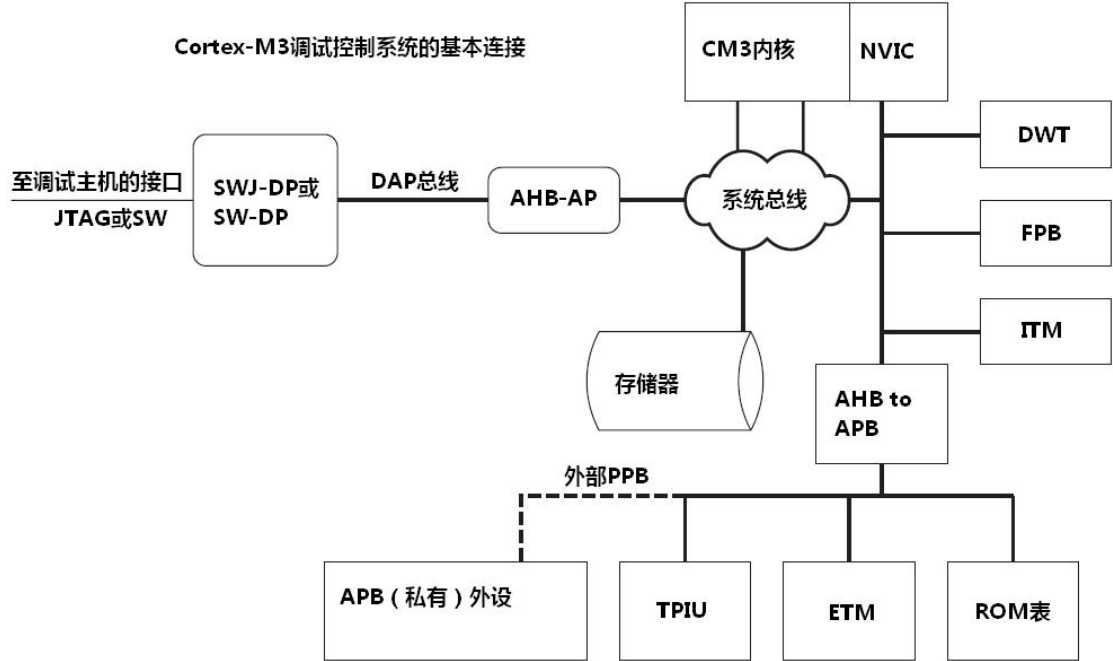
关于总线和总线桥的概念可以参考：

<https://zhuanlan.zhihu.com/p/400084595>

<https://blog.csdn.net/ZGQ760818970/article/details/78671791>



而在 CM3 中，调试设备共享同一个同一个存储器映射，如下图所示。



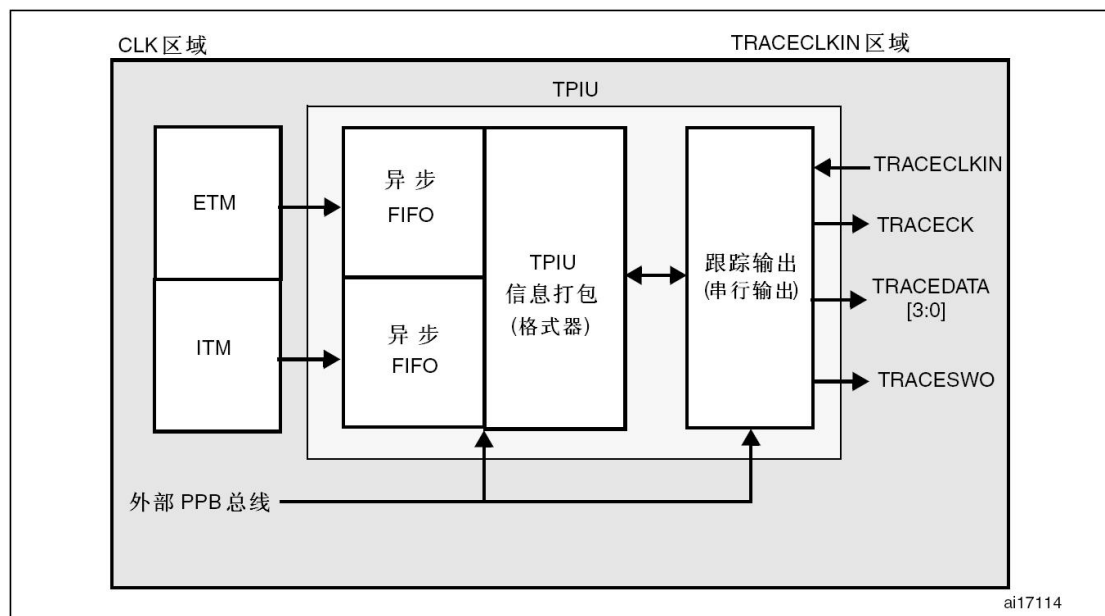
三、CoreSight 跟踪接口

CoreSight 跟踪接口 TPIU 在片上数据跟踪和跟踪宏单元之间担当桥梁的作用。

标准架构的 CoreSight 跟踪单元包括 ETM 嵌入式跟踪宏单元、PTM 程序跟踪宏单元、ITM 测量跟踪宏单元、HTM AHB 跟踪宏单元、STM 系统跟踪宏单元、ECT 嵌入式交叉触发、ETB 嵌入式跟踪缓存等。

每一款 MCU 都支持不一样的跟踪单元,而在 CM3 中只有 ETM 嵌入式跟踪宏单元和 ITM 测量跟踪宏单元,同时还有一个 DWT 数据观察点触发器。在跟踪过程中,由 ETM、ITM 或 DWT 产生的数据被裹成数据包,然后被送到“高级跟踪总线(ATB)”上进行传送。在 CoreSight 的架构中,如果某 SoC 含有多个跟踪源(例如多核系统),则需要一种硬件水平的 ATB 归并器(merger),把各 ATB 数据流归并成一条(在 CoreSight 架构中,这种硬件被名为 ATB funnel)。归并后的数据流都送往 TPIU(跟踪端口接口单元),TPIU 再把数据导出到片外的跟踪硬件设备。在数据送到了调试主机(PC)后,再由 PC 端的调试软件还原为先前的多条数据流。

以下是 STM32F1XX 系列单片机的 TPIU 框图。



四、调试功能的总结

嵌入式微处理器可以提供以下调试功能,以启用对应用程序的调试:

处理器状态修改 允许外部主机修改处理器状态,通过内部寄存器和内存系统决定内容。

处理器状态评估 允许外部主机访问内部寄存器和内存系统的内容,以评估处理器的运行状态。

可编程调试事件 允许外部主机对调试事件进行编程。外部主机通过配置调试逻辑,在发生特殊事件(例如,程序流到达代码中的特定指令)时,内核进入一种特殊的执行模式,在该模式下,内核状态可以在外部进行查看或修改。在本章中,这种特殊的执行模式被称之为调试状态。

进入或退出调试状态 允许外部主机决定处理器何时应进入或退出调试状态,或者强制处理器进入或退出调试状态。

跟踪特性: 跟踪与可编程事件相关联的程序流。

五、调试模式

在 CM3 中的调试操作模式分为两种：

第一种称为“halt”（停机模式），在进入此模式时，处理器完全停止程序的执行。

第二种则称为“debug monitor exception”（调试监视器模式），此时处理器执行相应的调试监视器异常服务例程，由它来执行调试任务，此时依然允许更高优先级的异常抢占它。调试监视器的异常号为 12，优先级可编程。除了调试事件可以触发异常外，手工设置其悬起位也可以触发本异常。

二者主要区别如下：

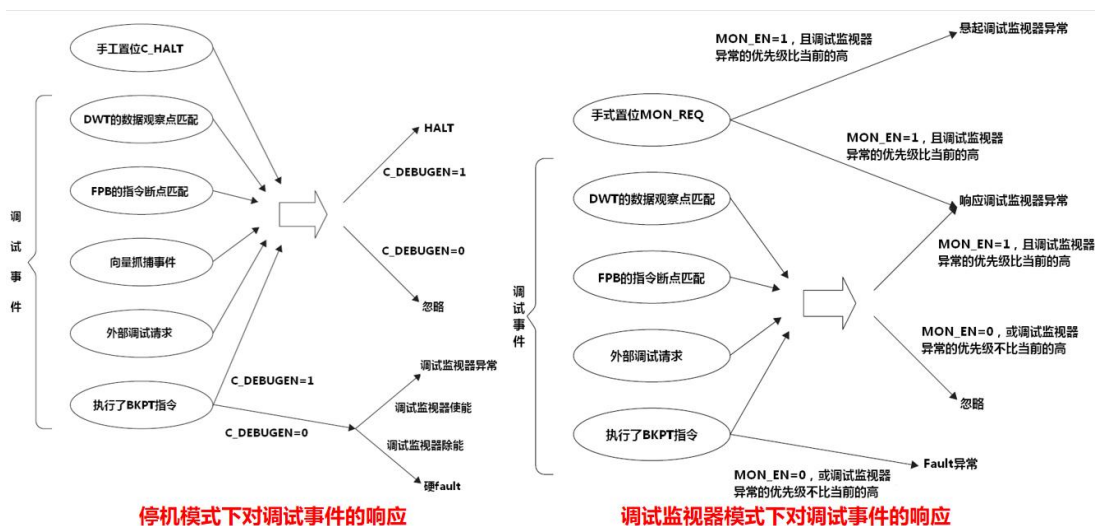
调试模式	特点
停机模式	<ol style="list-style-type: none"> 1. 指令执行被停止 2. systick 定时器停止 3. 支持单步操作 4. 中断可以在这期间悬起，并且可以在单步执行时响应，也可以屏蔽它们，使得单步时不受干扰
调试监视器模式	<ol style="list-style-type: none"> 1. 处理器执行监视器异常的服务例程（异常号： 12） 2. SysTick 定时器继续运行 3. 新来的中断按普通执行时的原则来抢占 4. 执行单步操作 5. 存储器的内容（如堆栈内存）会在调试监视器的响应前后更新，因为有自动入栈和出栈的操作

之所以加入调试监视器模式，是考虑到了在某些电子系统运行的过程中，是不可以停机的。例如，对于汽车引擎控制器以及电机控制器，就必须在处理调试动作的同时让处理器继续运行下去，这样才能保证被测试的设备不会意外损坏。有了调试监视器，就可以停止并调试线程级的应用程序以及低优先级的中断服务例程，同时高优先级的中断和异常能够响应。

六、调试事件

CM3 进入调试模式（停机模式和调试监视器模式）的来源有很多。

对于停机模式，满足下图 a 所示的条件可以喊停处理器。但即使是停机后，也可由上电复位和系统复位来复位处理器。在调试监视器模式下，满足下图 b 所示的条件可以喊停处理器。



图中，外部调试请求信号是通过 CM3 上的一个称为“EDBGREQ”的信号线传来的，该信号线的实际连接方式则取决于单片机/SoC 的设计。在有些场合下可以把该信号硬线连至低电平，从而永远无法发生；也可以把它连接到附加的调试组件上(芯片厂商可以添加额外的调试组件)：或者在多核系统中，可以用来连接其它处理机的调试事件。在调试活动结束后，通过清除 C_HALT 位，可以继续程序的执行。

在调试活动结束后，通过清除 C_HALT 位，可以继续程序的执行。

从图中可见，调试监视器模式下，与停机模式下的动作方式还是有一点区别的。这是因为调试监视器异常仅仅是异常的一种，它可以影响当前的优先级，但是不能使处理器停下来。

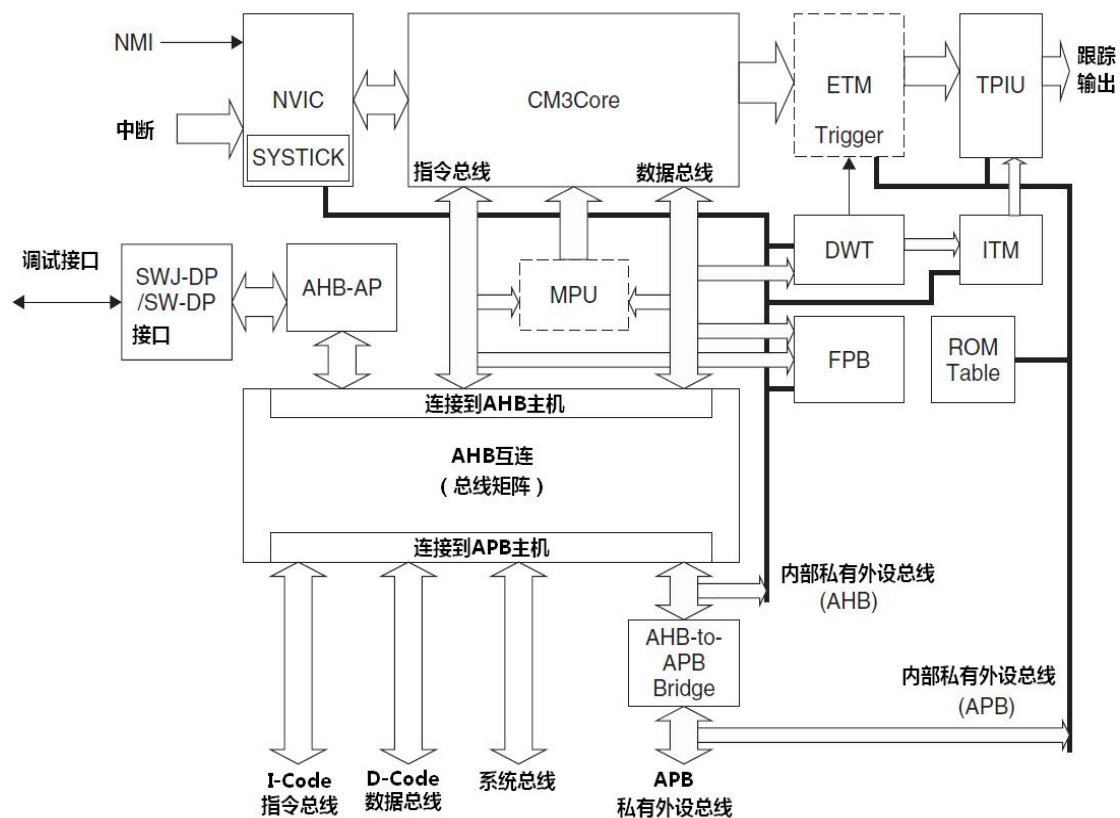
在调试活动结束后，通过该异常的返回，即可回到正常的程序执行中。

七、STM32 调试单元概况

STM32F10xxx 使用 Cortex™-M3 内核，该内核内含**硬件调试模块**，支持复杂的调试操作。**硬件调试模块允许内核在取指(指令断点)或访问数据(数据断点)时停止。内核停止时，内核的内部状态和系统的外部状态都是可以查询的。完成查询后，内核和外设可以被复原，程序将继续执行。**

当 STM32F10x 微控制器连接到调试器并开始调试时，调试器将使用内核的硬件调试模块进行调试操作。

下图为 STM32F10xxx 级别和 Cortex™-M3 级别的调试跟踪系统框图：



接下来我们从总体看一下 STM32F1 系列单片机的调试跟踪系统，包括以下部分：

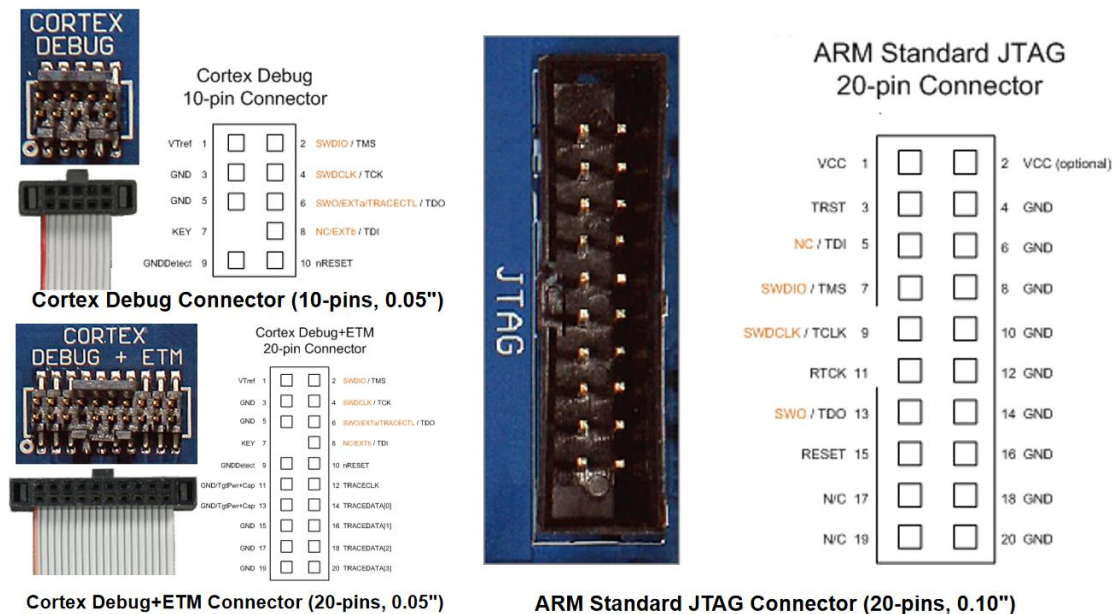
● **SWJ-DP：串行/JTAG 调试端口**

SWJ 调试端口(serial wire and JTAG)包括 JTAG-DP 接口(5 个引脚)和 SW-DP 接口(2 个引脚)。在 STM32F1 系列中,SW-DP 接口使用 PA13 和 PA14 两个引脚,JTAG-DP 接口使用 PA13、PA14 、PA15、PB3、PB4 五个引脚。其引脚作用和功能描述如下：

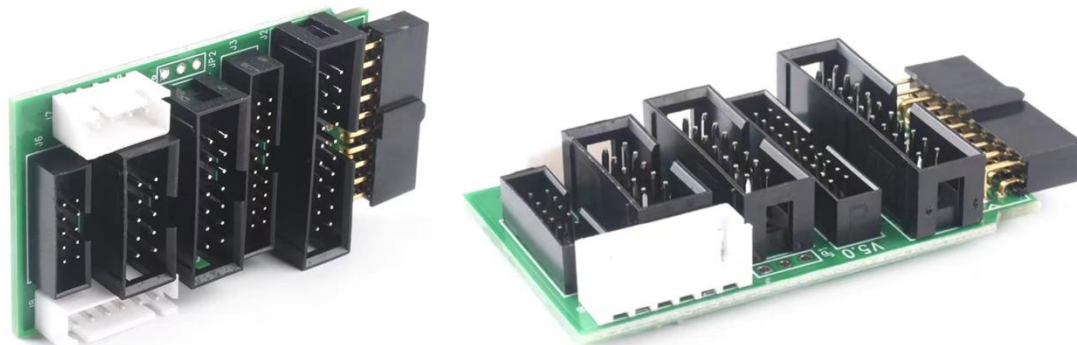
SWJ-DP端口引脚名称	JTAG 调试接口		SW 调试接口		引脚分配
	类型	描述	类型	调试功能	
JTMS/SWDIO	输入	JTAG模式选择	输入/输出	串行数据输入/输出	PA13
JTCK/SWCLK	输入	JTAG时钟	输入	串行时钟	PA14
JTDI	输入	JTAG数据输入	——	——	PA15
JTDO/TRACESWO	输出	JTAG数据输出	——	跟踪时为TRACESWO信号	PB3
JNTRST	输入	JTAG模块复位	——	——	PB4

单片机启动后，初始状态这五个引脚就默认是调试接口。而实际使用中，一般只使用 SWCLK、SWDIO 这两个引脚用做 SW 调试接口，其余的三个引脚可以空出来，重新定义为普通 I/O 来使用。如果需要将它们用作普通 IO 口，需要进行引脚复用，禁止调试功能。

这里，需要注意，虽然 JTAG 协议关键信号只有五种，但是支持 JTAG 协议的接口连接器是多种多样的，如下图所示：



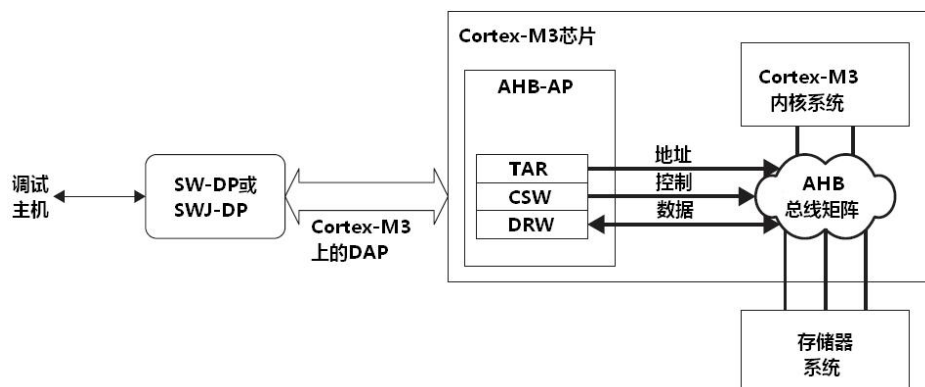
这个时候可以购买 JTAG 转接板进行连接。



● AHB-AP: AHB 访问端口

AHB-AP 位于 CM3 的存储器系统和调试接口模块 (SWJ-DP/SW_DP) 之间，充当一个总线桥的角色。对于大多数基本的在调试主机和 CM3 系统之间的数据传输，需要使用 AHB-AP 中的 3 个寄存器，它们是：控制及状态字 (CSW)、传输地址寄存器 (TAR)、数据读/写 (DRW)。

AHB - AP 的连接方法如下图所示：



● ROM 表:

CM3 的调试系统还包含了 ROM 表, 用于自动检测在某 CM3 芯片中包含了哪些调试组件。尽管作为 v7-M 的第一个践行者, CM3 拥有一个预定义的存储器映射并且包含了标准的调试组件, 但是新的 Cortex-M 器件可以包含不同的调试组件, 并且芯片厂商在实现 CM3 时也可以对调试组件加以修改。为使调试工具能检测到调试系统中具体包含的组件, 就提供了这张 ROM 表, 它记录了 NVIC 和各个调试功能块的地址。

ROM 表位于 0xE00F_F000。通过分析 ROM 表中的内容, 可以计算出系统和调试组件在存储器系统中的位置。在检测到了调试组件后, 调试器可以接下来查看它们的 ID 寄存器, 从而判定系统中哪些组件是可用的。

● ITM: 执行跟踪单元

ITM 是一应用驱动的跟踪源, 它支持 printf 类的调试手段来跟踪操作系统(OS)和应用事件, 并发布判定的系统信息。ITM 以包的形式发布跟踪信息, 它由以下部分组成:

- 软件跟踪: 软件可以通过直接写 ITM 激发寄存器来发布包信息。
- 硬件跟踪: ITM 会发布由 DWT 产生的信息包。
- 时间戳: 时间戳被发布到相应的包上。ITM 包含一个 21 位的计数器以产生时间戳。

CortexM3 的时钟或串行线观测器(Serial Wire Viewer)的位时钟率给计数器提供时钟。

由 ITM 发送的信息包输出到 TPIU(Trace Port Interface Unit), TPIU 再添加一些额外的包(参考 TPIU), 然后输出完整的包序列给调试器。用户在设置或使用 ITM 之前, 必需先使能异常调试和监视控制寄存器(Debug Exception and Monitor Control Register)的 TRCEN 位。

简单的讲, ITM 不需要暂停程序运行, 可以在程序全速运行的过程中实时输出变量的数值以便观察, 即 Trace 功能; 同时, 可以与 Printf 重定向相结合实现打印程序运行记录等功能而无需连接 USB 转 TTL 占用串口, 只需要 ITM 就可以输出调试信息。

需要特别注意的是, ITM 仅支持 SWD 调试, 而不支持 JTAG 调试, 原因在 STM32 参考手册中写明:

29.17.7 异步模式

调试模块提供一个低成本的, 只使用一个引脚的跟踪数据输出功能, 即使用异步输出引脚 TRACESWO。但显然, 这样的输出数据带宽是有限的。

TRACESWO 引脚与 JTDO 引脚复用, 只在 SW-DP 调试接口有效, 因此 STM32F10xxx 的所有封装都提供这种功能。

异步模式需要 TRACECLKIN 引脚有平稳的频率提供。对标准的 UART(NRZ)捕捉机制来说, 需要 5% 的正确度。曼彻斯特编码可放宽到 10%。

同时, 在 SWD 调试模式下, 调试器和单片机之间需要连接一个额外的引脚才能实现 ITM 功能:

● 异步模式

异步模式需要 1 个额外的引脚并且存在于所有的封装。此模式仅在串行调试接口有效(不支持 JTAG 调试接口)。

表212 异步跟踪引脚分配

TPIU引脚	同步跟踪模式		STM32F10xxx引脚分配
	类型	描述	
TRACESWO	输出	异步跟踪数据输出	PB3

这里, 还需要注意的是, CMSIS-DAP 调试器并不支持这个功能, 但是 J-LINK 和 ST-LINK 是可以的。

关于运用 ITM 实现数据跟踪和 Printf 输出的, 可以参考以下文章:

https://blog.csdn.net/malloc_luo/article/details/104374256

<https://blog.csdn.net/zhang062061/article/details/124071587>

<https://www.cnblogs.com/CodeWorkerLiMing/p/12007400.html>

同时 ITM 还附带了一个时间戳的功能：当一个新的跟踪数据包进入了 ITM 的 FIFO 时，ITM 就会把一个差分的时间戳数据包插入到跟踪数据流中。跟踪捕获设备在得到了这些时间戳后，就可以找出各跟踪数据之间的时间相关信息。另外，在时间戳计数器溢出时也会发生时间戳数据包。

● FPB：闪存指令断点

这里是 STM32 参考手册的介绍：

29.12 FPB (Flash patch breakpoint)

FPB单元：

- 实现硬件断点
- 用系统区域的代码和数据取代代码区域的代码和数据。此特性可以用来纠正代码区域内的软件错误。

软件补丁功能和硬件断点功能不能同时使用。

FPB由以下部分组成：

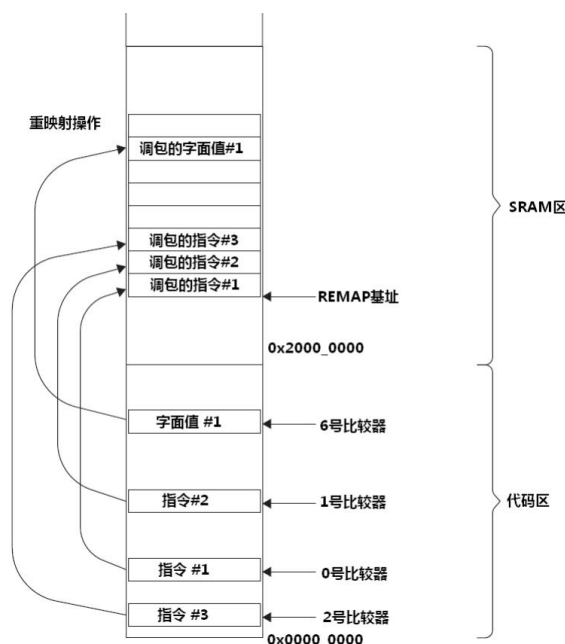
- 2个内容比较器，用来比较代码区域取得的内容并重映射到系统区域的相关地址。
- 6个指令比较器，用来比较代码区域的指令。这些比较器可用来实现软件补丁或者硬件断点功能。

FPB 闪存指令断点也称为地址重载及断点单元，主要提供 flash 地址重载和断点功能。

Flash 地址重载是指：当 CPU 访问的某条指令匹配到一个特定的 flash 地址时，将该地址重映射到 SRAM 中指定的位置，从而取指后返回的是另外的值。

使用这个重映射功能，可以创建一些“如果...将会...”(what if)形式的测试——通过把原始指令或字面值取代成另一个来实现，而且即使在 ROM 或 flash 中运行的代码，也能够参与此种测试。另一种用法很像“狸猫换太子”：对于某个位于 flash 中的子程序，在 SRAM 中提供一个冒充它的子程序。通过闪存地址重载，使得在执行到调用该子程序的指令(BL)时，实际上执行的是被“调包”过的，位于 SRAM 中的 BL，后者则跳转到“狸猫”中。这种机制使得基于 ROM 的设备也可以调试（修改过的子程序暂时放到 SRAM 中）。

下图演示了重映射的效果：



此外，匹配的地址还能用来触发断点事件，FPB 包含对应的 PC 地址比较器/指令地址比较器，当 CPU 想要运行对应地址的指令时，比较器会触发并返回 Breakpoint Instruction (BKPT) 指令给到 CPU，CPU 会暂停，从而实现硬件代码断点功能。

这里，关于不同代码断点类型可以看附二。

● DWT：数据观察点触发

(1) 数据观察点与指令断点

数据观察点与指令断点类似，指令断点使得我们根据正在执行的特定指令停止程序流，而数据观察点允许我们根据特定数据访问情况下停止程序流（观察点可设置为在数据“写”、“读”或“读/写”访问时产生中断）。例如，观察点可能配置为在更新变量、写入堆栈区域或读取特定缓冲区时产生中断。（想象一下这种情况：如果一段代码无意中改变了一个全局变量，而我们不知道这段代码是什么，因此无法设置指令断点。它可能是悬空指针、堆栈溢出或类似的东西，而我们无法使用代码断点进行追踪。这种情况下，数据观察点变成了不二选择）

数据观察点允许在某个内存单元被访问时停止 CPU，而不是在指令上中断/停止应用程序。观察点的数量和功能取决于所使用的微控制器。大多数供应商都提供了几个观察点，即使是一个观察点也可以为我们节省很多调试时间。通常，我们可以将其设置为在读取或写入访问时触发，或两者同时触发。数据观察点的相关硬件通常使用地址比较器来实现这一点：如果总线上的地址匹配，则 CPU 将停止。

(2) DWT 的组成单元和相关功能

DTW 可以产生数据观察点事件，让处理器进入调试模式，同时可以产生跟踪数据包（包括数据值、数据地址、当前 PC 值等信息），汇入到数据总线，经 Trace Port 输出。

DTW 主要由四个比较器组成，它们分别是：

①硬件观察点

在配置为发生比较匹配时，可以执行如下动作：

产生一个观察点调试事件，并且用它来调用调试模式，包括停机模式和调试监视器模式

②ETM 触发器

在配置为发生比较匹配时，可以执行如下动作：

可以触发 ETM 发出一个数据包，并汇入指令跟踪数据流中

③PC 值取样器

在配置为发生比较匹配时，可以执行如下动作：

程序计数器（PC）采样器事件触发

④数据地址取样器

在配置为发生比较匹配时，可以执行如下动作：

数据地址采样器触发

同时 DWT 内部还有一个时钟周期计数器 CYCCNT，CYCCNT 是一个向上的计数器，记录的是内核时钟运行的个数，内核时钟跳动一次，该计数器就加 1，精度非常高。通过 CYCCNT 可以完成对以下项目的计数：时钟周期、分支指令、存取单元操作、睡眠周期、CPI(每条指令的执行时间)和中断的额外开销。

当用于硬件观察点或 ETM 触发时，以上比较器既可以比较数据地址，也可以比较程序计数器 PC。当用于其它功能时，则只能比较数据地址。DWT 最常用的功能是使用 CYCCNT 寄存器来测量执行某个任务所花的周期数，或用作时间基准相关的目的（例如操作系统中统计 CPU 使用率），具体实现可参考：

<https://zhuanlan.zhihu.com/p/405212820>

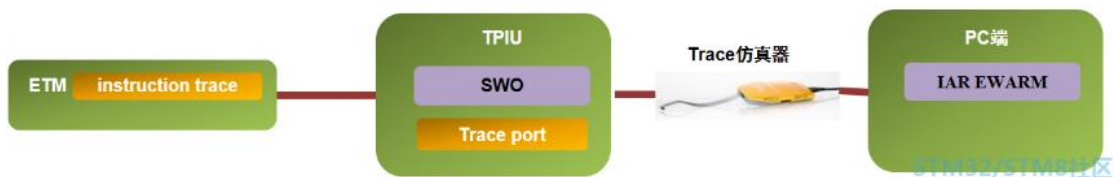
<https://blog.csdn.net/booksyhay/article/details/109028712>

● ETM：嵌入式跟踪微单元(在较大的封装上才有支持此功能的引脚)

在某些情况下，应用代码可能突然跑飞或者产生异常中断，此时导致问题的原因可能难以发现。调试这类问题时，需要使用 Trace 功能，其硬件实现基础为 ETM 功能块。

ETM 功能块用于提供指令跟踪（即指令执行的历史记录），它是个选配件，不一定出现在所有的 CM3 产品上。当它使能后，并且在跟踪操作开始后，它会产生指令跟踪数据包，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数据进行解析并还原出 MCU 内部的指令执行情况。

ETM 中也有一个 FIFO 缓冲区，为跟踪数据流的捕捉提供够用的时间。



为了减少产生的数据量，ETM 并不会一直精确地输出处理器当前正在执行的地址。通常它只输出有关程序执行流的信息，并且只有在需要时才输出完整的地址（例如，当一个跳转发生时）。因为调试主机也有一份二进制映像的拷贝，它可以使用此拷贝来重建指令的执行序列。ETM 也与其它调试组件互相交互。例如，它与 DWT 的比较器就有关系：DWT 的比较器可用于产生 ETM 的触发信号，或者控制跟踪的启动与停止。

这里需要注意，使用 Trace 功能时需要注意：

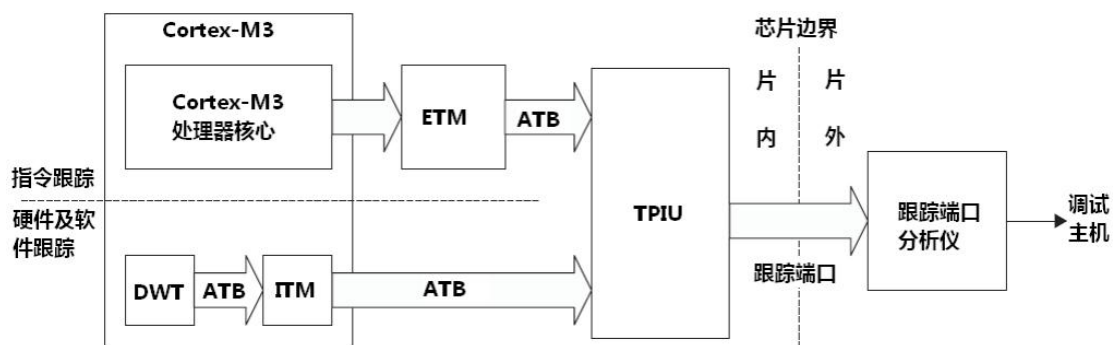
- (1) 芯片带有 ETM 模块
- (2) 仿真器支持 Trace 功能
- (3) Trace 数据输出引脚和调试接口连接

具体可以参考如下：

<https://shequ.stmicroelectronics.cn/thread-629340-1-1.html>

● TPIU：跟踪单元接口(仅较大封装的芯片支持)

如前所述，CM3 的跟踪系统是基于 CoreSight 架构的，跟踪数据被打成数据包，它们的长度可变。跟踪组件使用高级跟踪总线（ATB）来发送这些数据包给 TPIU，TPIU 则把它们格式化，转换成符合“跟踪总线接口协议”的数据包。格式化后的数据包发到片外，可以使用跟踪端口分析仪（TPA）之类的设备捕获它们。整个数据流动的路线如图所示：



CM3 的 TPIU 支持两种输出模式：

⌘带时钟模式(Clocked mode)，使用最多 4 位的并行数据输出端口

同步模式根据跟踪数据长度使用2到6个额外引脚，并且只存在于大封装芯片里。此外，此模式在JTAG调试接口和串行调试接口下都可使用，并提供比异步跟踪更好的数据输出量。

表213 同步跟踪引脚分配

TPUI引脚名	同步跟踪模式		STM32F10xxx引脚分配
	类型	描述	
TRACECK	输出	跟踪时钟	PE2
TRACED[3:0]	输出	同步跟踪数据输出，长度可以是1,2,或4	PE[6:3]

⌘串行线观察器（SWV）模式，使用单一位的 SWV 输出

异步模式需要1个额外的引脚并且存在于所有的封装。此模式仅在串行调试接口有效(不支持JTAG调试接口)。

表212 异步跟踪引脚分配

TPUI引脚	同步跟踪模式		STM32F10xxx引脚分配
	类型	描述	
TRACESWO	输出	异步跟踪数据输出	PB3

● MCU 调试模块(MCUDBG)

MCU调试模块协助调试器提供以下功能：

- 低功耗模式
- 在断点时提供定时器、看门狗、I²C和bxCAN的时钟控制
- 对跟踪脚分配的控制

八、SWV 调试

SWV 是由仪器化跟踪宏单元 ITM(Instrumentation Trace Macrocell)和 SWO 构成的。SWV 实现了一种从 MCU 内部获取信息的低成本方案，**SWO 接口支持输出两种格式的跟踪数据，但是任意时刻只能使用一种。两种格式的数据编码分别是 UART（串行）和 Manchester（曼彻斯特）**。当前 JLINK 仅支持 UART 编码，SWO 引脚可以根据不同的信息发送不同的数据包。当前 M3/M4/M7 可以通过 SWO 引脚输出以下三种信息：

(1) ITM 支持 printf 函数的 debug 调用（工程需要做一下接口重定向即可）。ITM 有 32 个通道，如果使用 MDK 的话，通道 0 用于输出调试字符或者实现 printf 函数，通道 31 用于 Event Viewer，这就是为什么实现 Event Viewer 需要配置 SWV 的原因。

(2) 数据观察点和跟踪 DWT(Data Watchpoint and Trace)可用于变量的实时监测和 PC 程序计数器采样。

(3) 时间戳, ITM 还附带了一个时间戳的功能: 当一个新的跟踪数据包进入了 ITM 的 FIFO 时, ITM 就会把一个差分的时间戳数据包插入到跟踪数据流中。跟踪捕获设备在得到了这些时间戳后, 就可以找出各跟踪数据之间的时间相关信息。另外, 在时间戳计数器溢出时也会发生时间戳数据包。

SWO 串行线输出是单引脚、异步串行通信, 可在 Cortex-M3/M4/M7 上使用, 并由主调试器探测支持。

SWO 输出, 需要一根 SWO (引脚) 线, 同时需要借助 SWV (查看器) 查看数据, 包括:

- 基于 Keil 的『Debug(printf)Viewer』
- 基于 IAR 的『Terminal IO』
- 基于 ST-LINK Utility 的『Serial Wire Viewer』
- 基于 J-Link 的『SWO Viewer』
- 基于 STM32CubeProg 的『Serial Wire Viewer』

使用 SWV 进行调试的相关教程可以参考:

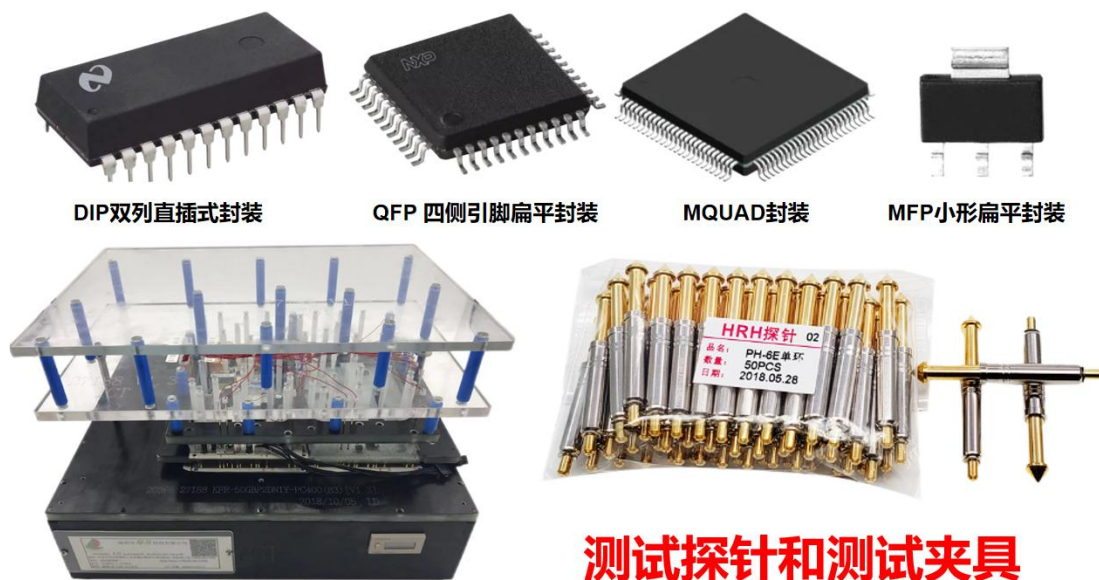
https://blog.csdn.net/Three_Future/article/details/124238309

<https://blog.csdn.net/ybhuangfugui/article/details/109554648>

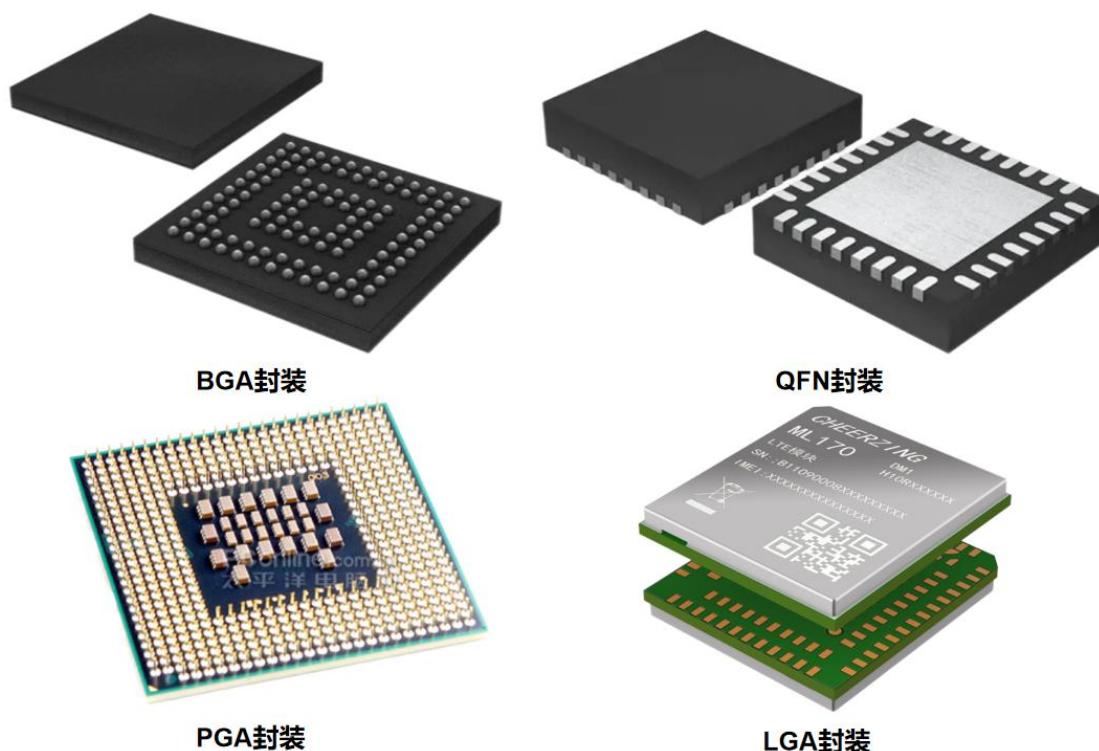
附一 JTAG 边界扫描的故事

考虑一下这个场景：有一个 PCB 板，上面包含了许许多多 IC 芯片，每个 IC 有许多引脚，各个引脚之间互有连接。不可避免的，由于焊接或制版工艺等种种原因，会产生相邻或本不该连接的引脚产生连接，那么我们如何知道呢？

一般来讲，我们会采用 ICT 即自动在线测试仪对电路板进行检测，ICT 测试可以检测的内容有：线路的开短路、线路不良、元器件的缺件、错件、元器件的缺陷、焊接不良等，并能够明确指出缺点的所在位置。对于下列引脚在侧边引出的芯片（如下图所示为常见封装）来说，我们可以使用测试探针配合测试夹具完成电路板的测量。



然而，对于以 BGA 球栅阵列封装形式的 IC 芯片来说，ICT 测试便十分难以开展。



同时，由于 ICT 测试价格高昂、测试周期长，同时许多电子工程师想在系统测试之前就能测试原型板卡的制造缺陷，甚至是在固件完成之前，于是一种用于测试、调试、编程和仿真印刷电路板的非侵入式方法—JTAG 应运而生。其优点如下：

更短的测试时间

对于小批量的板卡，总是很难证明测试夹具开发成本的合理性。在这些情况下，一个选择是飞针测试；但是这种技术的测试周期时间往往过长。JTAG 提供极短的测试时间并且不需要昂贵的夹具。

更低的测试开发成本

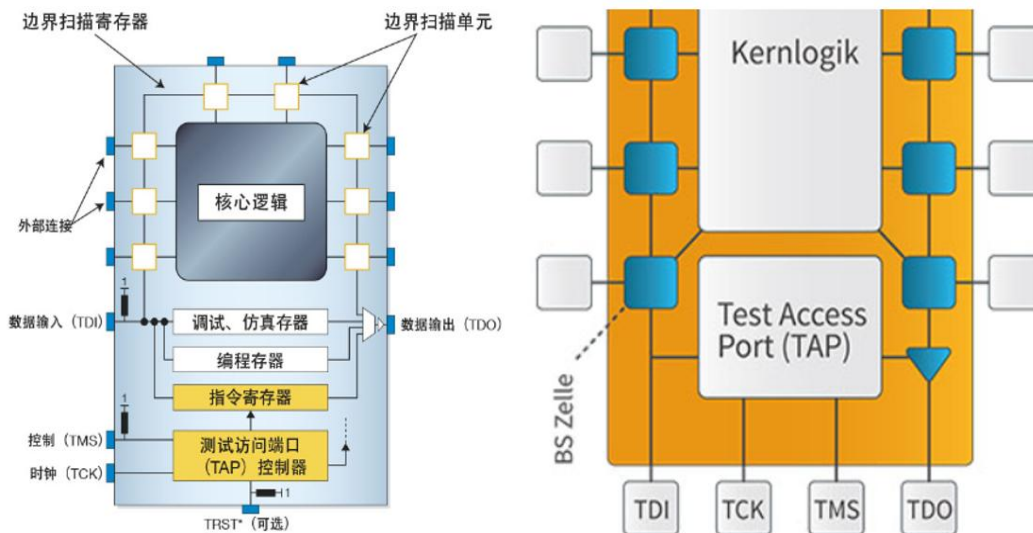
由于不同的处理器、FPGA 与外部设备交互的方式不同，传统的功能性测试需要高价的且针对每个板卡的定制开发。JTAG 大幅地降低了这类开发成本，因为它提供了一个简易的接口来控制用来与外围设备交互的 IO 针脚。这个标准接口，对于所有启用 JTAG 的设备都一样，意味着一套通用的测试模型可以在构建测试系统时被使用，并且重复使用。

测试、编程二合一的工具

还原连功能测试都无法运行的“死”板

如果板卡不能启动，那么传统功能测试就不能被运行；关键外围设备，例如 RAM 或者时钟上的简单故障都可以利用 JTAG 来找到，而功能测试则不能提供任何诊断信息

那么，JTAG 边界扫描是怎样被用来测试一个板卡的？



上图为边界扫描（JTAG 测试被称为边界扫描）的连接示意图，这里，TCK 和 TMS 两个信号以及可选项/TRST 信号都是广播信号；另一方面，朝向 TDO 的 TDI 形成串行链，即：扫描链或扫描路径（参见图示）。在模块级别，称之为测试总线。

这种结构的优点在于：无论有多少个组件切换到扫描链中，都不需要超过四条（包括可选项则为五条）信号线。

边界扫描单元（见上图）可以在两种模式中运行。在它的功能模式下，它并不影响设备的运行——这是板卡正常运行时所处在的模式。在它的测试模式下，它断开设备功能核心与针脚的连接。通过把边界扫描单元置于测试模式中，它可以被用来控制从一个被启用的设备向一个网里所驱入的值，而且还可以被用来监视那个网的值。

把对针脚的控制与被启用设备的功能断开明显地使得边界扫描测试开发比传统的功能测试更为容易，因为无论是设备置配，还是启动，都不需要使用针脚。通过提供一个从四针 TAP 控制和监视设备上所有被启用信号的机制，JTAG 显著地减少了测试板卡所需要的物理访问。

附二 代码断点类型

断点类型	实现原理	优缺点
硬件代码断点 Hardware Code Breakpoint	硬件代码断点是通过 MCU 内部的专门硬件模块来实现。以 ARM Cortex-M 为例, FPB 包含对应的 PC 地址比较器, 当 CPU 想要运行对应地址的指令时, 比较器会触发并返回 BKPT 指令给到 CPU, CPU 会暂停, 从而实现硬件代码断点功能。	硬件代码断点的优势是速度快(因为是通过 MCU 内部的硬件比较器实现), 劣势是数量有限 (ARM Cortex-M 最多是 6-8 个)。
软件代码断点 Software Code Breakpoint	软件代码断点是通过调试器 (Debugger) 将对应地址的指令替换成专门的断点指令。以 ARM Cortex-M 为例, 调试器将对应地址的指令替换成 BKPT 指令。当 CPU 想要运行对应地址的指令时, 将会运行 BKPT 指令, CPU 会暂停, 然后对应地址的指令又会恢复成原来的指令, 从而实现软件代码断点功能。	软件代码断点的优势是数量没有限制, 但是由于对应地址的指令需要被调试器替换成专门的断点指令, 只能用于在 RAM 中运行的代码。
闪存代码断点 Flash Code Breakpoint	闪存代码断点是 IAR Systems 公司在 IAR Embedded Workbench for ARM version 7.60.1 上新加的一个功能, 通过使用 IAR Systems 公司的 I-jet 硬件调试器, 可以在一些基于 ARM Cortex-M 的 MCU 上设置没有数量限制的闪存代码断点。目前, J-Link 同样支持无限闪存断点, 它允许用户在闪存中调试时设置无限数量的断点。J-Link 的“无限闪存断点”适用于内部和外部闪存, 甚至内存映射的 QSPI 闪存。	优点有二: 其一, 断点数量无限; 其二, J-Link 的闪存代码断点可以在硬件断点不能使用的情况下也能工作, 例如外部存储器或存储器映射的 QSPI flash 超出了可以设置硬件断点的区域。在大多数 Cortex-M3 和 M4 设备上, 硬件断点不能用于外部存储器, J-Link 的闪存代码断点可以。 但是由于闪存代码断点需要在下载的时候使用 Flash Loader 来擦除和重写 Flash, 只能在调试模式开始之前设置。

关于不同断点类型, 可以参考 <https://stackoverflow.com/questions/8878716> 下问题回答

What is the difference between hardware and software breakpoints?

Asked 11 years, 4 months ago Modified 1 year, 3 months ago Viewed 57k times

在 CM3 中, 最常用的有两种断点机制: 断点指令和基于由 FPB 地址比较器的硬件断点。

参考资料

MCU: ARM 的调试架构

<https://zhuanlan.zhihu.com/p/24517355>

无需昂贵的仿真器而能够调试 51 单片机

<https://blog.csdn.net/czleclub/article/details/5767507>

CH55X 在线仿真方法说明——使用 ISD51 进行在线仿真

<https://www.wch.cn/bbs/thread-65921-1.html>

User's Guides for Keil C51 Development Tools

<https://developer.arm.com/documentation/101655/latest/>

Cortex™-M3(r1p1 版)技术参考手册(TRM)

<https://download.csdn.net/download/skd2278/1483760>

ARM 调试接口 V5

<https://download.csdn.net/download/wxiangxiang/12746369>

ARM CoreSight 开发工具集(r1p0 版)技术参考手册

https://developer.arm.com/documentation/ddi0480/b/DDI0480B_coresight_soc_r1p0_trm.pdf

Coresight (一) coresight 简介

https://blog.csdn.net/qq_37573794/article/details/121645434

【ARM Coresight 介绍 3 - ARM Cortex-M DWT】

https://blog.csdn.net/sinat_32960911/article/details/127772979

【ARM Coresight 及 DS-5 介绍 1 - Coresight 介绍】

https://blog.csdn.net/sinat_32960911/article/details/124352860

STM32 强大的调试和跟踪 CoreSight 技术

<https://blog.csdn.net/ybhuangfugui/article/details/105445449>

JTAG 的 TAP 状态机介绍

<https://blog.csdn.net/chenyuheu/article/details/117035639>

What is JTAG?

<https://www.corelis.com/education/tutorials/jtag-tutorial/what-is-jtag/>

JTAG 接口定义与其他简介

<https://blog.csdn.net/beetleinv/article/details/86372466>

AN4989 Application note STM32 microcontroller debug toolbox

jtag 接口电路设计 (jtag 接口有哪些功能)

<https://www.eolink.com/news/post/66493.html>

JTAG 1 - What is JTAG?

<https://www.fpga4fun.com/JTAG1.html>

什么是 JTAG 那么,我怎样才能利用它呢?

<https://www.xjtag.com/zh-hans/about-jtag/what-is-jtag/>

[译文] TAP and TAP Controller // JTAG 测试访问接口及其控制器

<https://zhuanlan.zhihu.com/p/598667697>

【JTAG】1149.1 协议详解

<https://blog.csdn.net/xuhe0206/article/details/125867462>

边界扫描如何工作? 什么是 JTAG?

<https://www.goepel.com/zh>

Cortex-M3 权威指南

芯片封装类型大全

<https://zhuanlan.zhihu.com/p/142449536>

Arm® Debug Interface Architecture Specification ADIv6.0

<https://developer.arm.com/documentation/ih0074/latest/>

ARM 架构上用来替代 JTAG 的调试协议 SWD

<https://www.4hou.com/posts/MEg1>

ARM 调试接口——PART B.4 SWD 协议解析

<https://zhuanlan.zhihu.com/p/196389730>

一文帮你彻底搞懂 ARM Debug Interface 之 SWD

https://www.sohu.com/a/604654567_121124018

SWD 下载器通信协议底层原理

https://mp.weixin.qq.com/s/_gPurOVfyTq07Js0d9YOKA

调试备忘录-SWD 协议解析

<https://blog.csdn.net/xue745146527/article/details/117404897>

ARM 调试接口——PART A 概述

<https://zhuanlan.zhihu.com/p/349680641>

Embedded Trace Macrocell Architecture Specification ETMv1.0 to ETMv3.5

<https://developer.arm.com/documentation/ih0014/latest/>

接口与协议学习笔记-AMBA 片上通信协议_APB_AHB_AXI_AXI4 不同版本（二）

<https://www.cnblogs.com/ui0jhi/archive/2018/07/31/9366884.html>

ARM 调试 CoreSight、ETM、PTM、ITM、HTM、ETB 等常用术语解析

<https://www.myir-tech.com/customerService/resource-list.asp?id=510>

嵌入式跟踪单元 ETB MTB (Micro Trace Buffer)的实现

<https://blog.csdn.net/u014285530/article/details/105687411>

Zephyr 的博客

<https://www.cnblogs.com/lvzh/p/14314535.html>

代码调试跟踪与优化（二）--- 如何调试嵌入式代码？

https://blog.csdn.net/m0_37621078/article/details/114918430

在 IAR Embedded Workbench 中有效地使用不同类型的代码断点

<https://zhuanlan.zhihu.com/p/552597296>

Debugger flow control: Hardware breakpoints vs software breakpoints

<http://www.nynaeve.net/?p=80>

J-Link Unlimited Flash Breakpoints

<https://www.segger.com/products/debug-probes/j-link/technology/flash-breakpoints/>

Faster Debugging with Watchpoints

<https://interrupt.memfault.com/blog/cortex-m-watchpoints>

Watchpoints: Data Breakpoints

<https://mcuoneclipse.com/2012/04/29/watchpoints-data-breakpoints-in-mcu10/>

使用 IAR ETM Trace 调试功能的要求

<https://shequ.stmicroelectronics.cn/thread-629340-1-1.html>

调试相关名词：JTAG，SWD，SWO，ITM，SWV，RTT，Event Recorder 解释

<https://www.armbbs.cn/forum.php?mod=viewthread&tid=110921>

基于 STM32CubeProg、[Serial Wire Viewer] SWO 打印输出

<https://blog.csdn.net/ybhuangfugui/article/details/109554648>

【Serial Wire Viewer (SWV) 】

https://blog.csdn.net/Three_Future/article/details/124238309