第16章

# 调试组件

* 简介
* 跟踪组件：数据观察点与跟踪(DWT)
* 跟踪组件：仪器化跟踪宏单元(ITM)
* 跟踪组件：嵌入式跟踪宏单元(ETM)
* 跟踪组件：跟踪端口接口单元(TPIU)
* 闪存地址重载与断点单元(FPB)
* AHB访问端口
* ROM表

## 16.1 简介

在CM3的大礼包中有很多调试组件，使用它们可以执行各种调试功能：断点、数据观察点、闪存地址重载以及各种跟踪等。如果您是一位软件开发人员，则也许永远无需了解调试组w件的细节，因为它们通常只是由调试器及其周边工具使用的。

本章对每种调试组件做一个基本的介绍，如果需要了解它们的更详细信息，如编程模型，则请参阅《Cortex-M3 Technical Reference Manual(Ref1)》。

所有的调试及跟踪组件，以及FPB，都可以经由CM3的私有外设总线来编程。在大多数情况下，只有调试主机才会编程这些组件。强烈反对应用程序尝试访问调试组件（除了对ITM中stimulus端口寄存器的访问），这样做很容易与调试器发生冲突。

### 16.1.1 Cortex-M3的跟踪系统

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，并且它们的长度可变。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16.1所示：

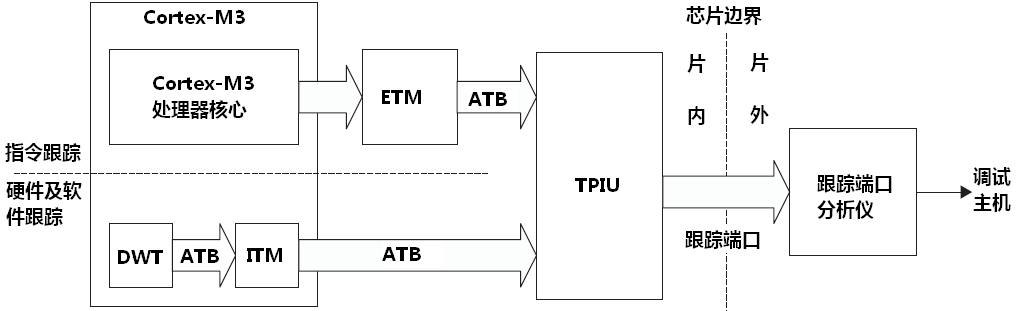


图16.1 Cortex-M3的消化系统模式图

从上图可见，在CM3中可以有3种跟踪源：ETM, ITM和DWT。其中，ETM是一个可选组件，因此有些CM3芯片中没有配。在操作中，每个跟踪源都被赋予一个7位的ID号（ATID），随着它所发出的数据包一起送出。这样，在从归并的数据流中还原各原始的数据流时，就可以使用ATID来作为识别的手段。与其它标准的CoreSight组件不同的是，CM3的调试组件内建了归并ATB数据流的逻辑；而在标准的CoreSight系统中，ATB数据包归并器是一个独立的功能块，并且被称为“ATB funnel”。

在使用跟踪系统之前，必须把DEMCR.TRCENA置位（回顾表15.2，或者参阅表D.37）。在这之前，跟踪系统是处于除能状态的。在正常的操作中，如果不需要跟踪，则通过清零TRCENA来除能一些与跟踪有关的逻辑，可以降低系统的功耗。

## 16.2 跟踪组件：数据观察点与跟踪(DWT)

本节的主角是DWT，它提供的调试功能包括：

1. 它包含了4个比较器，可以配置成在发生比较匹配时，执行如下动作：
   1. 硬件观察点（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
   2. ETM触发，可以触发ETM发出一个数据包，并汇入指令跟踪数据流中
   3. 程序计数器（PC）采样器事件触发
   4. 数据地址采样器触发
   5. 第一个比较器还能用于比较时钟周期计数器（CYCCNT），用于取代对数据地址的比较
2. 作为计数器，DWT可以对下列项目进行计数：
   1. 时钟周期（CYCCNT）
   2. 被折叠（Folded）的指令
   3. 对加载/存储单元（LSU）的操作
   4. 睡眠的时钟周期
   5. 每指令周期数（CPI）
   6. 中断的额外开销（overhead）
3. 以固定的周期采样PC的值
4. 中断事件跟踪

当用于硬件观察点或ETM触发时，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，比较器则只能比较数据地址。

每一个比较器都有3个寄存器

* COMP寄存器
* MASK寄存器
* FUNCTION控制寄存器

其中，COMP寄存器是一个32位寄存器，用于存储要比较的值。MASK寄存器可以用于掩蔽数据地址的一些位，被掩蔽的位不参与比较。如表16.1所示：

表16.1 MASK寄存器定义

|  |  |
| --- | --- |
| MASK | 被忽略的位段 |
| 0 | 忽略所有的位 |
| 1 | 忽略[0] |
| 2 | 忽略[1:0] |
| 3 | 忽略[2:0] |
| … |  |
| 15 | 忽略[14:0] |

比较器的FUNCTION寄存器用于决定该比较器的功能。为了避免潜在的不可预料的行为，必须先编程MASK和COMP，最后再编程RUNCTION。如果要更改某个比较器的功能，必须先把FUNCTION清零——除能该比较器，再重新配置一回，依然是最后配置FUNCTION。

DWT中有剩余的计数器，它们典型地用于程序代码的“性能速写”（profiling）。通过编程它们，就可以让它们在计数器溢出时发出事件（以跟踪数据包的形式）。最典型地，就是使用CYCCNT寄存器来测量执行某个任务所花的周期数，这也可以用作时间基准相关的目的（操作系统中统计CPU使用率可以用到它）。

## 16.3 跟踪组件：仪器化跟踪宏单元（ITM）

ITM有如下的功能：

* + 软件可以直接把控制台消息写到ITM stimulus端口，从而把它们输出成跟踪数据。
  + DWT可以产生跟踪数据包，并通过ITM把它们输出。
  + ITM可以产生时间戳数据包并插入到跟踪数据流中，用于帮助调试器求出各事件的发生时间。

因为ITM要使用跟踪端口来输出数据，所以芯片上必须有TPIU单元，否则无法输出——在使用ITM前要确认此事。如果不幸地没有TPIU，也还可以使用NVIC调试寄存器，或者使用最后一招——求助于UART来输出控制台消息。

欲使用ITM，必须把DEMCR.TRCENA位置位，否则ITM处于除能状态，无法使用。

另外，在ITM寄存器中还有一个锁。在编程ITM之前，必须写入一个访问钥匙值0xC5AC\_CE55（CoreSight的ACCESS）到这个解锁寄存器。否则，所有对ITM寄存器的写操作都被忽略。

最后，ITM自己也是另一个控制寄存器（可能是说控制寄存器的名字也是“ITM”吧），用于控制对各功能的独立使能。

控制寄存器中含了ATID位段，作为ITM在ATB中的ID值。这个ID必须是唯一的——每个跟踪源都必须有唯一的ID值，从而使调试主机能从接收到的跟踪数据包中分离出各跟踪源的数据。

### 16.3.1 基于ITM的软件跟踪

ITM的一个主要用途，就是支持调试消息的输出（例如，printf格式的输出）ITM包含了32个刺激(stimulus)端口，允许不同的软件把数据输出到不同的端口，从而让调试主机可以把它们的消息分离开。通过编程“跟踪使能寄存器”，每个端口都可以独立地使能/除能，还可以允许或禁止用户进程对它执行写操作。

与基于UART的文字输出不同，使用ITM输出不会对应用程序造成很大的延迟。在ITM内部有一个FIFO，它使写入的输出消息得到缓冲。不过，为了安全起见，最好还是在写入前检查该FIFO被填满的程度。

输出的消息被送往TPIU，然后可以通过“跟踪端口接口”或者“串行线接口”来收集它们。在最终的代码中也无需移除产生调试消息的代码，而是可以把TRCENA位清零，这样ITM就被除能，调试消息也不会输出，你也可以在一个“live”系统中开启消息输出。另外，通过设置跟踪使能寄存器，可以限定允许使用的端口。

### 16.3.2 基于ITM和DWT的硬件跟踪

ITM也能用于输出硬件跟踪数据，这些数据由DWT产生，ITM则担任跟踪数据包的归并单元，如图16.2所示。欲使用DWT跟踪，需要在ITM控制寄存器中置位DWTEN位，剩下的DWT跟踪设置在DWT中完成。

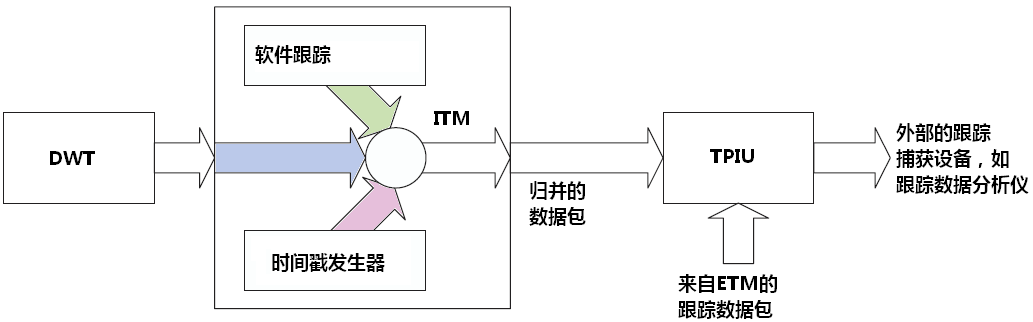


图16.2 在ITM和TPIU上的数据包归并模式图

### 16.3.3 ITM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。

## 16.4 跟踪组件：嵌入式跟踪宏单元

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。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并不会一直忙不迭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也与其它的调试组件互相交互。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。

与传统ARM处理器的ETM不同的是，CM3的ETM没有自己的地址比较器，而是由DWT的比较器代为完成。事实上，CM3的ETM与传统ARM的ETM有很大的区别。

欲使用ETM，必须执行下述的建立步骤（由调试器及其周边工具完成）

1. 把DEMCR.TRCENA位置位（DEMCR寄存器的定义参见表15.2或D.37）。
2. 解锁ETM以编程它的寄存器：往ETMLOCK\_ACCESS寄存器中写0xC5AC\_CE55。
3. 编程ATBID寄存器（ATID），赋予ETM一个唯一的标识，以便把它的跟踪数据包与其它跟踪源的跟踪数据包分开。
4. ETM的NIDEN输入信号必须为高电平。该信号的实现是取决于具体的器件的，还需要参考该器件的数据手册。
5. 编程ETM控制寄存器组以产生跟踪数据。

## 16.5 跟踪组件：跟踪端口接口单元（TPIU）

如前所述，ITM, DWT和ETM的跟踪数据都在TPIU处汇聚。TPIU用于把这些跟踪数据格式化并输出到片外，以供跟踪端口分析仪之类的设备接收使用。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
  + 串行线观察器（SWV）模式，使用单一位的SWV输出（不适用于早期版本的CM3）

在带时钟模式下，数据输出端口实际使用的位数是可编程的。这取决于两点。其一，是芯片的封装；其二，是在应用中，提供了多少个信号引脚给跟踪输出使用。在具体的芯片中，通过检查TPIU的寄存器，可以判断跟踪端口的最大尺寸。此外，跟踪数据输出的速度也是可编程的。

在SWV模式下，则使用SWV协议。它减少了所需的输出信号数，但是跟踪输出的最大的带宽也减少了。

欲使用TPIU，需要先把DECMR.TRCENA置位，还要编程“协议选择寄存器”和“跟踪端口尺寸寄存器”，这个工作由跟踪捕捉软件完成。

Cortex-M3 r2p0修订版

在SWV模式下，会使用SWV协议。这时，输出信号就只需要1个比特了，但是跟踪输出的最高带宽也会下降。另外，在使用串行线调试协议时，SWV模式的输出可以和TDO共享信号线。这样一来，哪怕使用只带有标准JTAG接口的入门级调试器，也可以通过DWT和ITM来捕捉跟踪信息。

## 16.6 闪存地址重载及断点单元（FPB）

FPB有两项功能：

* 硬件断点支持。产生一个断点事件，从而使处理器进入调试模式（停机或调试监视器异常）
* 把代码地址空间中对指令或字面值(literal data)的加载，重载到SRAM的地址空间中。

FPB有8个比较器，分别是：

* + 6个指令比较器
  + 2个字面值比较器

### 什么是“字面值加载”？

当我们使用汇编写程序时，常常需要往寄存器中加载立即数据。当立即数的值很大时，加载操作就无法用单一指令完成，例如：

LDR R0, =0xE000E400

因为没有任何指令能接收32位立即数，我们需要把这个立即数预先安置到另一个存储器空间中，通常放到程序代码区的后面，然后就可以使用一条相对PC的加载指令，来读取这个立即数到对应的寄存器中。因此，上条代码的汇编结果可以如下所示：

LDR R0, [PC, #<immed\_8>\*4]

; immed\_8 = (字面值地址 – PC)/4

...

; 文字池

...

DCD 0xE000E400

...

上面的LDR也可以是Thumb-2提供的32位版本：

LDR.W R0, [PC, #+/-<offset\_12>]

; offset\_12 = 字面值地址-PC

...

; 文字池

...

DCD 0xE000E400

...

在实际使用中我们经常需要在代码中使用多个字面值，汇编器或编译器就会在代码区中开出一块地址范围，来集合字面值，这个块就是所谓的“文字池”。在CM3中，从文字池的数据加载通常使用D-Code总线，但比较另类的实现也可以把文字池放到RAM区中，从而使用系统总线加载。

在FPB中有一个闪存地址重载控制寄存器，它包含了FPB的使能位。此外，每个比较器在它自己的控制寄存器中，都还有各自的使能位——前者是总开关。两种使能位必须都为1时才能启用比较器。

可以通过编程比较器，把指令空间的地址重载（重映射）到SRAM地址空间中。当使用此功能时，需要编程REMAP寄存器，以提供需要重映射内容的基址。REMAP寄存器的最高3位[31:29]被硬线连接成0b001，因此限定了重映射后的地址范围在0x2000\_0000-0x3FFF\_FF80之间，这段地址正好落在SRAM地址空间中。

当指令地址或字面值地址与比较器中的数值发生匹配命中时，读访问就会根据REMAP的设置被重映射。

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。并且即使是在ROM或flash中运行的代码，也能够参与此种测试。另一种用法在本质上与这种用法相同，但被取代的是跳转指令，因此行为很像“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。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

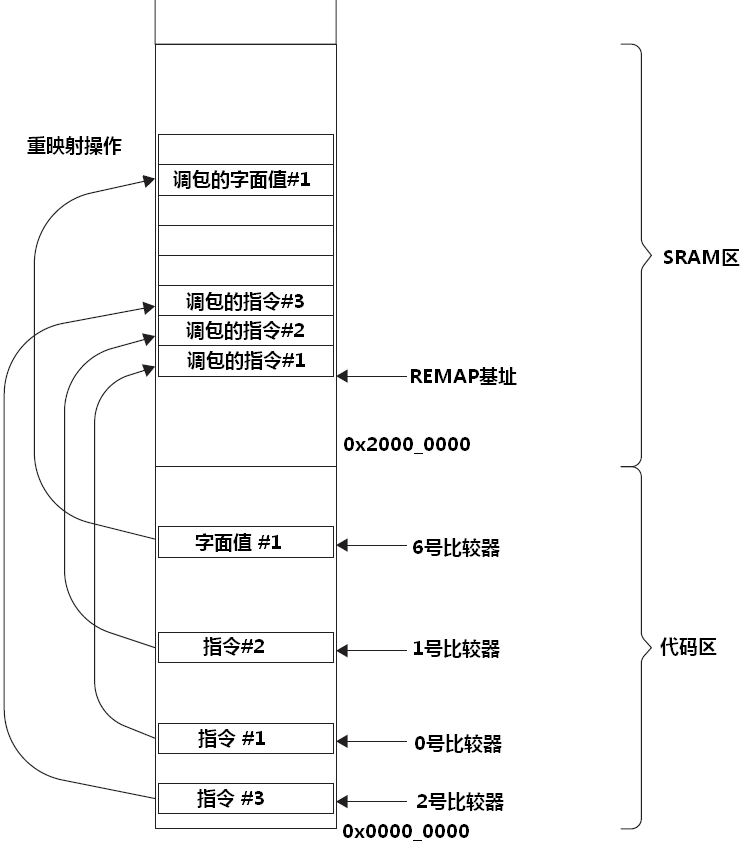


图16.3 闪存地址重载：对指令及字面值的重映射

除了地址重载，指令地址比较器的另一项功能，就是用于产生硬件断点（共6个），当地址匹配时使处理器进入调试模式。

## 16.7 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，只需要使用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的连接方法如图16.4所示：

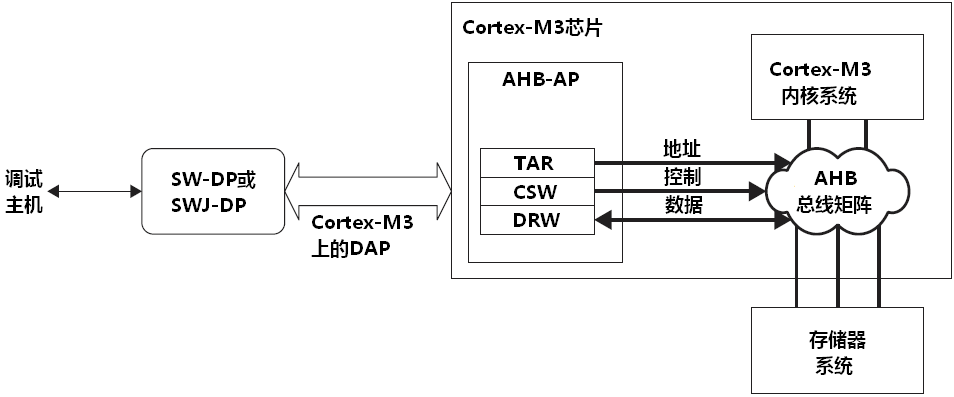


图16.4 在Cortex-M3中AHB-AP的连接

CSW寄存器可以控制传送方向（读/写）、传送大小以及传送类型等。TAR寄存器则指令传送地址，而DRW寄存器则容纳了被传送的数据（在访问该寄存器时就启动了传送）。DRW中的数据与总线上实际显示的是一致的，所以对于半字和字节传送，必须由调试硬件把得到的数据适当移位，以对齐到LSB。例如，若欲在地址0x1002上执行一次半字传送，则需要把数据放到DRW的[31:16]上。AHB-AP可以产生非对齐传送，但是它不会根据地址偏移来自动对目标数据做圆圈移位，必须由调试软件堵上这个窟窿：要么手工圆圈移位，要么把未对齐访问分解为若干个对齐的访问。

在AHB-AP中还有其它的寄存器，它们提供附加的功能。例如，AHB-AP中提供了4个bannked寄存器和地址自动增量的功能，用于加快在小范围连续地址中数据访问的速度。

在CSW寄存器中，还有一个名为MasterType的位。通常需要把它置1，以此告知参与AHB-AP数据传送的硬件：该数据传送是调试器发起的。但是，调试器也可以清零此位来伪装成处理器内核。这样，在AHB上接收数据的硬件就会以为是内核发起的数据传送，从而正常地动作。这个功能可以用于测试目的，尤其是对于带有FIFO的外设，用于获知当它被调试器访问时，行为有什么不同。

## 16.8 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。

在CM3的ROM表中，第一条目的内容应当是：NVIC的入口地址相对于ROM表入口地址的偏移量。ROM表首条目的缺省值是0xFFF0F003，其中位段[1:0]的作用比较特殊：它指示本条目对应的设备是存在的，并且在本条目的后面还有后续的条目（也就是说本条目不是最后一个条目）。这样，通过第一个条目，我们就知道系统中有NVIC，并且还有第2个条目，而且还能计算出NVIC的地址为0xE00F\_F000+0xFFF0\_F000=0xE000\_E000。

缺省的ROM表如图16.2所示。但是因为芯片厂商可以添加、移除以及把某些可选的组件替换成其它的CoreSigth调试组件，这时该芯片的ROM表就会与缺省的有所不同，以反映出相应的变化。

表16.2 Cortex-M3缺省的ROM表

|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 数值 | 名称 | 功能 |
| 0xE00F\_F000 | **0xFFF0\_F003** | **NVIC** | **指向NVIC的基址：0xE000\_E000** |
| 0xE00F\_F004 | **0xFFF0\_2003** | **DWT** | **指向DWT的基址：0xE000\_1000** |
| 0xE00F\_F008 | **0xFFF0\_3003** | **FPB** | **指向FPB的基址： 0xE000\_2000** |
| 0xE00F\_F00C | 0xFFF0\_1003 | ITM | 指向ITM的基础： 0xE000\_0000 |
| 0xE00F\_F010 | 0xFFF4\_1003/  0xFFF4\_1002 | TPIU | 指向TPIU的基址： 0xE004\_0000 |
| 0xE00F\_F014 | 0xFFF4\_2003  0xFFF4\_2002 | ETM | 指向ETM的基址： 0xE004\_1000 |
| 0xE00F\_F018 | 0 | End | End-Of-Table标记 |
| 0xE00F\_F0CC | 1 | MEMTYPE | 表示在此存储器映射中，可以访问系统存储器 |
| 0xE00F\_F0D0 | 0 | PID4 | 外设ID空间，保留 |
| 0xE00F\_F0D4 | 0 | PID5 | 外设ID空间，保留 |
| 0xE00F\_F0D8 | 0 | PID6 | 外设ID空间，保留 |
| 0xE00F\_F0DC | 0 | PID7 | 外设ID空间，保留 |
| 0xE00F\_F0E0 | 0 | PID0 | 外设ID空间，保留 |
| 0xE00F\_F0E4 | 0 | PID1 | 外设ID空间，保留 |
| 0xE00F\_F0E8 | 0 | PID2 | 外设ID空间，保留 |
| 0xE00F\_F0EC | 0 | PID3 | 外设ID空间，保留 |
| 0xE00F\_F0F0 | 0 | CID0 | 组件ID空间，保留 |
| 0xE00F\_F0F4 | 0 | CID1 | 组件ID空间，保留 |
| 0xE00F\_F0F8 | 0 | CID2 | 组件ID空间，保留 |
| 0xE00F\_F0FC | 0 | CID3 | 组件ID空间，保留 |

数值的最低两个位用于指示该设备是否存在(bit[1])以及后面还有没有其它的表项(bit[0])。在正常情况下，NVIC, DWT和FPB总是必须存在的，因此最后两位永远是1。然而，TPIU和ETM则可以被裁掉，并且可能被CoreSight家庭中其它的调试组件所取代。

数值的高位部分用给出对应组件的入口地址相对于ROM表入口地址的偏移量。例如，

NVIC入口地址= 0xE00F\_F000 + 0xFFF0\_F000 = 0xE000\_E000（进位位被忽略）

在开发调试工具时，有必要从ROM表中一一查兑各调试组件，因为难免会有些另类的CM3芯片会自定义调试组件，并且修改ROM表，而通过计算ROM表得到的地址是可以拿去拍板的。