Application Note

C2000™ 硬件内置自检

Salvatore Pezzino, Peter Ehlig and Whitney Dewey

摘要

本应用手册讨论 C2000™ 实时控制器中的硬件内置自检 (HWBIST) 功能。HWBIST 提供了一种在 C28x CPU 上达到高水平诊断覆盖率的方法，这通常是满足安全标准所必需的。

内容

插图清单

表格清单

商标

1 引言

HWBIST 是指由 ATPG 工具生成的电路和扫描模式，用于筛查目标电路内的逻辑故障。这种方法广泛用于半导体器件测试。所有 C2000 器件在器件制造过程中都利用某种级别的硬件辅助测试。一些较新的 C2000 器件支持客户在其系统测试中使用此测试技术，以测试 CPU 的完整性 (US 8,799,713 B2)。本文档介绍如何以及为何在系统级别利用 HWBIST。

表 1-1 列出了本应用报告中使用的术语和缩写。

表 1-1. 术语和缩写

缩写	术语
ATPG	自动生成测试模式
BIST	内置自检
捕获	当种子通过待测逻辑计时时，嵌入式电路捕获变化逻辑的结果
C28x	TMS320C28x 器件中内核 32 位 CPU 的名称
CS	Code Composer Studio™
CLA	控制律加速器
环境恢复	在完成硬件 BIST 微运行后恢复中央处理器 (CPU) 寄存器和状态标志的过程。这是由软件执行的。
环境保存	在启动硬件 BIST 微运行之前保存 CPU 寄存器和状态标志的过程。这是由软件执行的。
内核界限	在微运行测试期间，CPU 内核与外设和中断信号断开连接。测试后，内核重新连接到这些信号。
覆盖率	硬件 BIST 所覆盖的 CPU 逻辑的百分比。
CPU	中央处理器
CRC	循环冗余校验
DC	诊断覆盖率
闪存	非易失性片上存储器
FPU	浮点单元
HWBIST	硬件内置自检
ISR	中断服务例程
JTAG	联合测试行动组。JTAG 是一种基于扫描的通信协议（如 I2C），它允许扫描到测试电路或仿真电路。
微运行	执行完整 HWBIST 测试执行的一部分。HWBIST 设计为支持分段执行完整覆盖率测试，以更好地管理中断延迟和功耗。这些微运行必须在更小的时间片中执行，以实现更高效的任务调度。在微运行期间，CPU 与所有外设和存储器隔离。此外，中断由 HWBIST 控制器记录。
MISR	多输入签名寄存器
NMI	不可屏蔽中断
PEST	定期自检
PLL	锁相环
POR	加电复位
POST	开机自检
RAM	随机存取存储器
ROM	只读存储器
种子	使用扫描路径加载到电路中的初始状态，以便电路在测试开始之前以已知状态开始
信标	通过多内核器件上使用的 CPU1 或 CPU2 获取对某些自检寄存器的写访问权限的机制
SDL	软件诊断库
STL	自检库
TMU	三角函数加速器
TRM	技术参考手册
VCU	Viterbi 和复杂数学单元

在具有 HWBIST 的 C2000 器件上，HWBIST 以 C28x CPU 和 FPU、VCU、CRC 以及 TMU 加速器为目标。还包括仿真分析电路，该电路管理这些处理元件和仿真器之间的通信，以及管理诸如断点、观察点和单步执行等功能。

HWBIST 不针对器件上的其余逻辑。器件上的另一个逻辑可通过器件特定安全手册中所述的其他自检或诊断机制进行测试。

有关 HWBIST 的可用性，请参阅器件特定数据表。您还可以查看以下文档，以了解 HWBIST 以及其他功能安全特性和配套资料：

- C2000™ 实时微控制器的汽车功能安全特性
- C2000™ 实时微控制器的工业功能安全特性

虽然本文档重点介绍 HWBIST，但对于没有 HWBIST 且符合功能安全要求的 C2000 器件（如 F28004x 器件系列中的此类器件），提供了一个 C28x 自检库 (C28x STL)，以使用基于软件的方法测试 CPU 逻辑的完整性。同样，HWBIST 不包括控制律加速器 (CLA)，因此，对于具有 CLA 的器件，可以使用单独的 CLA 自检库 (CLA STL)。这些库是应要求提供的，请联系您当地的 TI 销售人员以请求访问权限。

1.1 HWBIST 概述

图 1-1 展示了如 C2000 器件中使用的 HWBIST 方框图。橙色和粉红色部分展示了用于测试的逻辑。在系统使用中，这个逻辑是系统代码的处理引擎。数据按照执行系统代码的指示流经锁存器。

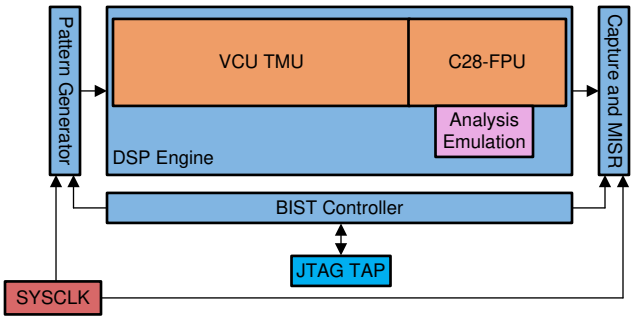

图 1-1. HWBIST 方框图

然而，这些相同的锁存器包括扫描访问，以便在测试该逻辑期间，高速测试流可以验证电路中门的操作。在这里讨论的情况下，逻辑中有许多并行扫描路径，以便可以并行测试逻辑的各重要部分。在这种测试模式下，逻辑的运行方式与处理器在运行代码时不同。

图形发生器为这些并行扫描路径提供种子，以提供在逻辑上验证目标门的操作所需的活动。这些种子是计算机生成的，并且使用标准 ATPG 工具验证覆盖率。这些种子经过优化，可在极少的周期内满足特定故障等级目标。这些优化器的供应商对这种优化感到非常自豪。

备注

这种优化意味着晶体管的开关速率明显高于该逻辑执行系统代码时发生的开关速率。此外，这提供了非常高的故障覆盖率。

捕获和 MISR 部分跨所有并行链获取扫描操作的结果。扫描模式通过路径的步进交互与连接到锁存器的电路中的其他逻辑门交互。优化软件将故障注入到门中，如果 MISR 没有识别出故障，则需要额外的种子来验证出现故障的门。优化器将获得一个覆盖率目标，并将继续生成种子，直到满足此指标。达到 60% 的覆盖率相对简单；达到 95% 需要明显更多的种子，而达到 99% 则需要比 95% 明显更多的种子。

备注

当通过并行扫描路径驱动位时，所有目标锁存器的环境会多次更改。换句话说，在测试之前这些锁存器中的任何环境都会在测试期间完全丢失。可通过硬件逻辑和软件的组合来恢复环境。

扫描操作的计时由 SYSCLK 驱动。BIST 控制器管理数据在扫描流程中如何移位和计时。BIST 控制器还管理 MISR 的种子和比较值的加载。在器件制造测试流程中，BIST 控制器和时钟源是使用 JTAG 等器件测试端口建立的。

这是对该测试方法的非常简化的描述。网络上提供了许多关于基于扫描的测试的详细学术文章。

1.1.1 HWBIST 系统内运行

如前所述，一些 C2000 器件支持使用 HWBIST 来筛查 CPU 在系统中是否存在逻辑故障，而不仅仅是在器件制造测试期间。图 1-2 展示了一个方框图，其中包括支持此选项的附加电路。

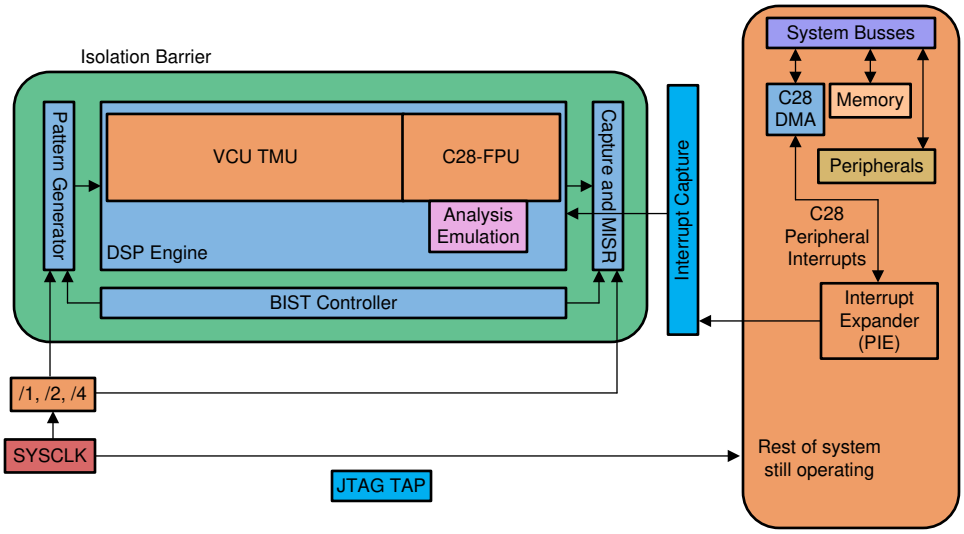

图 1-2. 系统内 HWBIST 方框图

在器件测试环境中使用 HWBIST 时，测试仪管理每个目标逻辑部分以提供器件的总体测试。然而，当器件在系统中时，系统的各方面可能会受到待测目标逻辑的活动的不利影响。这些方面包括器件上和器件外部。出于这个原因，HWBIST 在目标逻辑周围包含一个屏障，以便 HWBIST 测试的活动与系统的其余部分隔离。在微运行测试期间，CPU 与外设和中断信号断开连接。测试后，内核重新连接到这些信号。这称为*内核界限*。在图 1-2 中，隔离屏障显示为绿色。同样，待测逻辑必须与系统中其他地方的活动隔离。这一屏障也发挥同样的作用。

但是，如果系统必须引起待测 CPU 的注意，那么它可能提供中断。当 BIST 控制器释放目标逻辑时，这些中断被捕获在缓冲器中，并提供给待测 CPU 逻辑。目标逻辑的完整覆盖率测试需要一段时间。覆盖率目标越高，所需时间就越长。C2000 器件中的 BIST 控制器将总覆盖率分为若干小部分来执行和验证，尽可能减少这些捕获的中断的延迟。这也解决了通过并行扫描路径产生的较高晶体管开关速率的一些功率问题。

在极端情况下，系统资源可能生成一个 NMI，该 NMI 会停止 HWBIST 操作并使用 HWBIST 复位恢复待测 CPU 运行。一旦环境恢复完成，就会获取 NMI 矢量，并且 NMI 服务例程可以解码 NMI 标志寄存器以确定中断的来源。NMI 将在 HWBIST 软件返回到调用序列之前设置陷阱。用户应用程序必须相应地管理 NMI 响应。

器件制造 HWBIST 和系统内 HWBIST 之间的一个显著差异在于：器件测试仪通过测试端口与 BIST 控制器进行通信，而系统内 HWBIST 使用 CPU 与 BIST 控制器进行通信。HWBIST 正在测试的 CPU 是管理 HWBIST 控制器的 CPU。更具体地说，待测 C28x CPU 控制 BIST 操作，其中所有锁存器都被多次更改。

下面是这个过程的工作原理。在 CPU 上运行的代码执行以下操作：

1. 初始化 HWBIST 中的操作模式，映射 CPU 复位以响应 HWBIST 返回到服务例程（该例程专门编码用于从 HWBIST 返回）。
2. 打开中断捕获缓冲器。
3. 保存 CPU 和相关联的基于代码的加速器的环境。
4. 启动 HWBIST 执行的小时间片。此时 CPU 不再是 CPU，而是开始成为待测逻辑。
5. 在完成 HWBIST 执行的小时间片段后，HWBIST 控制器：
   - 在状态寄存器中捕获结果。如果检测到逻辑故障，BIST 控制器会向 CPU 生成 NMI。
   - 对 CPU 逻辑生成 CPU 复位：
     - 此复位将 CPU 置于一个已知和受控的状态。
     - 释放此复位后，待测逻辑再次变为 CPU。
6. CPU 执行 HWBIST 复位服务例程：
   - 恢复保存的环境
   - 关闭 HWBIST 控制器操作的剩余部分
   - 释放存储在中断捕获缓冲器中的中断
   - 返回到可以在其中读取 HWBIST 状态寄存器的调用序列

当 HWBIST 正在执行时，系统的其他方面也可以运行。如前所述，来自片外或片上源的中断保存在中断捕获缓冲器中。但是，将在 HWBIST 主动测试 CPU 时处理映射到 DMA 通道的触发器。例如，一旦完成步骤 6，就可以收集来自 SCI 或 I2C 端口的系统相关命令，并将其从端口移动到要处理的系统存储器中。这是因为所有器件总线都使用 HWBIST 屏障进行隔离。

前面概述中列出的所有操作都在 C2000 软件诊断库 (SDL) 中执行，该库是 C2000Ware 的一个组件。SDL 提供功能安全软件机制的实现，可帮助系统开发人员实现其安全目标。有关应用程序代码如何可以调用提供的 HWBIST 驱动程序的详细信息，请参阅 SDL 软件包中包含的用户指南。SDL 用户指南是有关 HWBIST 的器件特定的详细信息的更佳来源，例如支持的诊断覆盖率和执行时间。请在 `<C2000Ware install directory>/libraries/diagnostic/<device>/docs` 中查找此文档。

1.2 HWBIST 故障响应

如前所述，当 C28x CPU 启动 HWBIST 控制器时，CPU 会关闭，以便 HWBIST 引擎可以测试内部的逻辑。图 1-3 在状态图中展示了此操作的流程。

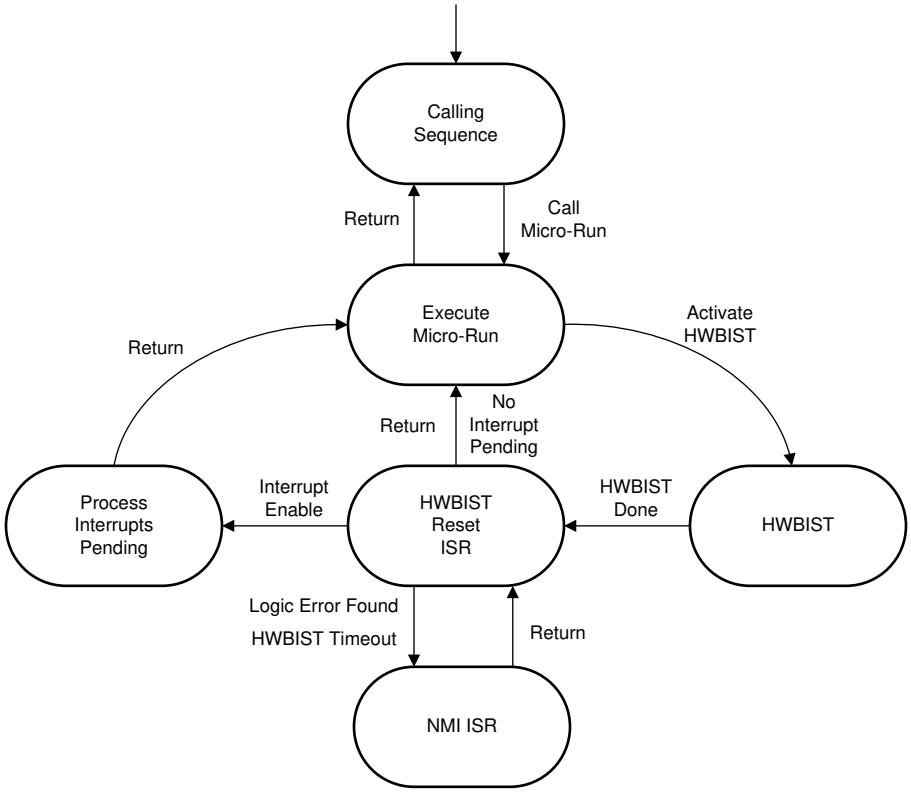

图 1-3. HWBIST 状态图

当 HWBIST 检测到故障时，它会设置 HWBIST 状态寄存器中的相应位并退出 HWBIST 运行。此错误可能以如下形式出现：

- 检测到逻辑故障
- HWBIST 控制器在没有完成微运行的情况下超时

在任何一种情况下，HWBIST 控制器都将故障信息保存到 HWBIST 状态寄存器中，为处理器生成 NMI，并在 NMI 标志寄存器中设置适当的位。在双处理器器件中，HWBIST 控制器为每个处理器生成 NMI。

1.3 使用系统内 HWBIST 的优点

在系统中使用 HWBIST 的原因有很多。下面展示了五个合理的示例：

- 在系统的初始设计验证和调试期间，验证 C2000 器件在系统中是否正确连接。

  HWBIST 对于原型调试的这一方面可能过于严格。仿真器为这项工作提供了更简单的方法。
- 验证 C2000 器件在连接到电路板后是否仍然正常工作。

  作为系统制造的一部分，了解器件在电路板制造过程中没有受到损坏是很有用的。电路板制造事件极有可能对器件造成灾难性损坏，在这种情况下，HWBIST 无法在系统内运行。此外，损坏极有可能发生在未经 HWBIST 测试的引脚驱动器/缓冲器、外围电路或嵌入式存储器上。电路板或系统制造事件极不可能只损坏 HWBIST 所针对的电路。在电路板制造过程中，器件发生损坏的情况并不常见。但是，如果器件受到损坏，最好尽早知晓，以便在电路板生产线上进行调整。
- 当器件在系统中正常工作后，检查器件是否受到损坏。器件损坏极有可能是由于以下原因之一造成的：
  - 上电期间过载
  - 断电期间过载
  - 电源事件导致的电压过载
  - 温度过载

  在系统启动时运行 HWBIST 可解决前两个原因。系统温度和电压监控器可解决其余两个原因
- 监控器件是否避开了制造测试。

  这并不是 HWBIST 在系统中的有效用途，因为 HWBIST 已经在器件测试仪环境中运行，在该环境中可以通过更高的裕度（电压和温度）来执行。然而，如果 HWBIST 确实捕获了故障，则有理由担心系统中的某个器件在数据表中定义的工作范围之外运行良好。这可能无法在器件的引脚上测量，因为它可能是一个瞬时事件。
- 监控器件是否有降级机制。

  使用此电路时，晶体管出现一定程度的降级是正常的，符合预期。这是个小问题，设计和器件测试包括补偿这种漂移的裕度。

  此外，还有一些潜在缺陷无法通过正常的器件测试方法进行筛查。这些缺陷机制要求施加一定程度的压力来加速故障发生过程。在器件制造测试中使用压力测试来加速大多数此类降级缺陷机制。

  最后，HWBIST 有助于识别这些避开主动器件制造测试的降级机制。

2 使用系统内 HWBIST

本节介绍如何在系统中使用 HWBIST 以及与 C2000 软件诊断库紧密结合。本节仅提供有关如何执行 HWBIST 的摘要，但在 C2000Ware 的《SDL 用户指南》中提供了更多详细信息。本节还详细介绍在双核器件上运行的注意事项和调试技巧。

备注

在 C2000 软件诊断库中发布的用于配置和执行 HWBIST 的软件通常不应由用户修改。虽然可以对 NMI 处理程序或错误标志管理进行细微调整，但更改 HWBIST 初始化和整体执行流程可能会导致诊断覆盖率级别低于记载的级别。

2.1 基本 HWBIST 操作

执行 HWBIST 涉及以下四个代码段：

- 初始化 HWBIST 控制器
- 执行 HWBIST
- 从 HWBIST 中恢复
- 管理结果

大部分过程是通过软件诊断库函数实现的，用户的应用程序可以调用这些函数。头文件 stl_hwbist.h 中提供了函数定义。有关这些函数描述的更多详细信息，请参阅 stl_hwbist.h 或特定于器件的《SDL 用户指南》（位于库发布包的 /docs 文件夹中）。

包含多达八个函数，如下所示：

```
__interrupt void STL_HWBIST_errorNMIISR(void);
uint16_t STL_HWBIST_runFull(const STL_HWBIST_Error errorType);
uint16_t STL_HWBIST_runMicro(void);
void STL_HWBIST_restoreContext(void);
void STL_HWBIST_init(const STL_HWBIST_Coverage coverage);
void STL_HWBIST_injectError(const STL_HWBIST_Error errorType);
bool STL_HWBIST_claimSemaphore(const STL_HWBIST_Core core);
void STL_HWBIST_releaseSemaphore(void);
```

2.1.1 初始化 HWBIST 控制器

初始化 HWBIST 控制器是通过调用这个库函数来完成的：

```
void STL_HWBIST_init(const STL_HWBIST_Coverage coverage);
```

此函数初始化 HWBIST 控制器以进行操作。*coverage* 参数为枚举类型 STL_HWBIST_Coverage 并指定要实现的覆盖率。如果在多核器件上，则此函数预计尝试在 HWBIST 上执行运行的 CPU 已声明 HWBIST 信标。此函数初始化 HWBIST 寄存器，如下所示：

- 针对第一个 <= 95% 覆盖率的每次微运行的周期数，以及针对增量覆盖率达到 99%（如果该器件支持）的每次微运行的周期数：
  - 尽可能地减小微运行的时间片
  - 尽可能地减小环境延迟
  - 尽可能地减小微运行期间的功耗
- HWBIST 时钟配置
- 返回地址为 0x0000，即存储器的 RAMM0 块的开头
- 由输入参数 *coverage* 指定的覆盖率级别

2.1.2 执行 HWBIST

HWBIST 执行许多微运行操作，直到满足完全覆盖。软件诊断库提供了两个选项来完成 HWBIST。第一个选项是在每次 HWBIST 完整运行时，使用 STL_HWBIST_init() 对 HWBIST 控制器执行一次初始化，然后定期执行 STL_HWBIST_runMicro()，直到 HWBIST 完成。此选项允许对 HWBIST 进行更小的时间分片。第二个选项是调用 STL_HWBIST_runFull()，它完成一次完整的 HWBIST 运行，然后返回到用户的代码。此选项需要更长的时间，对于开机自检或可以分配更多时间来执行完整 HWBIST 运行时更有用。

2.1.2.1 执行 HWBIST 微运行

要在待测 CPU 内核已声明信标并已执行一次性初始化后执行 HWBIST 的一次微运行，必须调用以下函数：

```
STL_HWBIST_runMicro();
```

此函数执行待测 CPU 的 HWBIST 微运行，并返回微运行的状态。此函数设计用作定期自检 (PEST)。

图 2-1 展示了一个流程图，该图详细介绍了 STL_HWBIST_runMicro() 函数的设计。 《诊断库用户指南》中也提供了此信息。

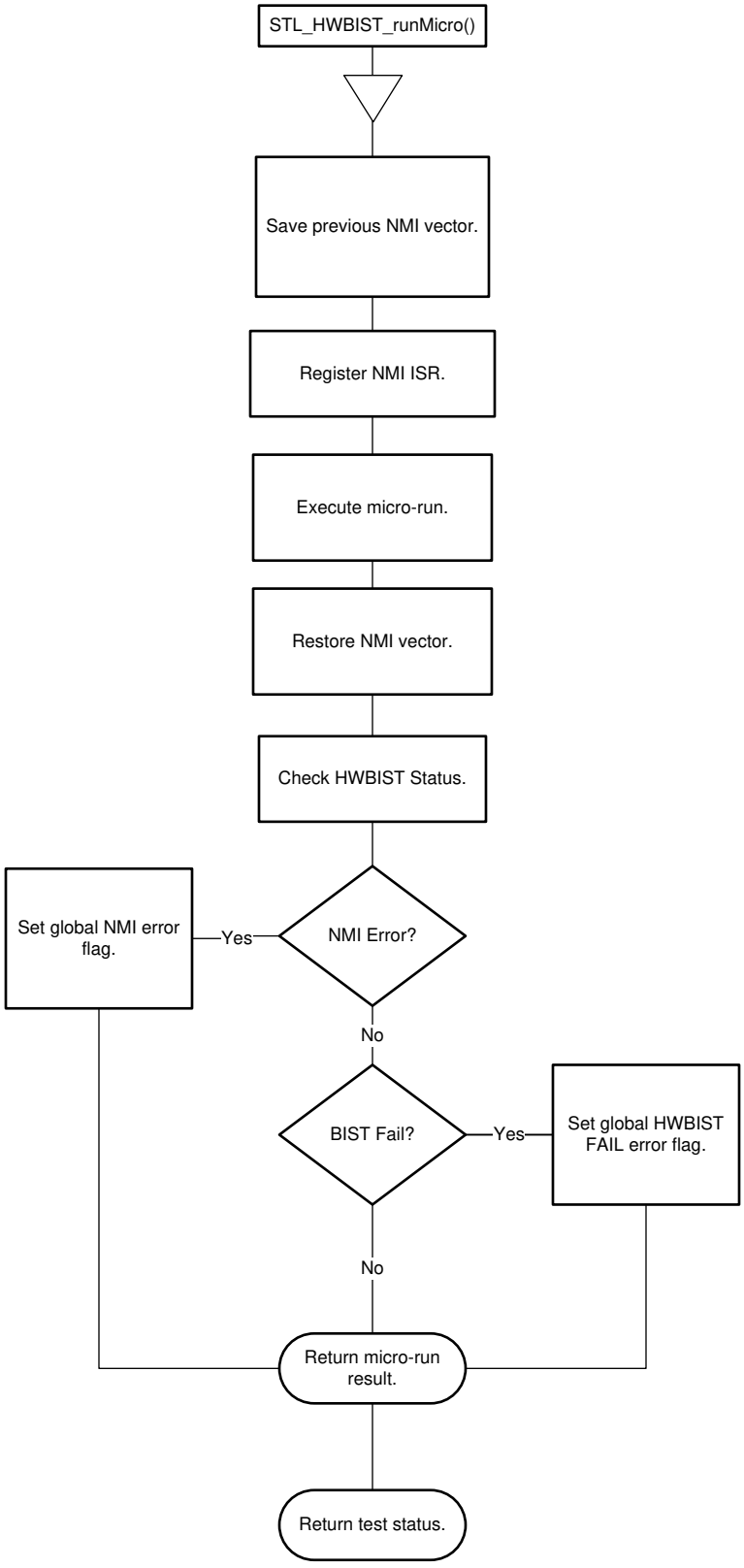

图 2-1. STL_HWBIST_runMicro() 流程图

要使用 STL_HWBIST_runMicro() 函数针对小于或等于 95% 的覆盖率级别执行完整的 HWBIST，必须执行以下函数序列：

1. 如果在多核器件上，请声明 HWBIST 信标。

```
STL_HWBIST_claimSemaphore();
```

2. 根据器件上可用的选项，将 HWBIST 初始化为 95% 或更低的覆盖率。

```
STL_HWBIST_init(STL_HWBIST_95_LOS);
```

3. 执行 HWBIST 微运行。

```
STL_HWBIST_runMicro();
```

4. 重复步骤 3，直到完成或观察到错误。执行 STL_HWBIST_runMicro()，直到它完成且没有错误，或者直到通过返回值观察到错误、设置全局错误标志或触发 NMI。跟踪调用 STL_HWBIST_runMicro() 的次数。如果对于覆盖率级别达到预期的微运行次数之后，返回值仍然不指示 HWBIST 已完成，则认为这是一个超时错误。微运行次数在代码中定义为 stl_hwbist.h 中的 *STL_HWBIST_MICRO_LIMIT_<coverage>*。例如，在 F28002x 上，STL_HWBIST_MICRO_LIMIT_90 设置为 750，这意味着要达到 90% 的诊断覆盖率，应用程序需要调用 STL_HWBIST_runMicro() 750 次。
5. 如果在多核器件上，请释放 HWBIST 信标。

```
STL_HWBIST_releaseSemaphore();
```

要使用 STL_HWBIST_runMicro() 函数执行覆盖率为 99% 的完整 HWBIST (仅在 F2837x、F2807x 和 F2838x 器件上受支持) ，您通常按照上述步骤以达到 95%，重新初始化 HWBIST 以达到 99% 的覆盖率，然后重复调用 STL_HWBIST_runMicro()，直到它完成或发现错误。这些步骤因器件不同而稍有差异。有关更多详细信息，请参阅器件特定的《SDL 用户指南》。

图 2-2 展示了单个时间分片的微运行的设置和执行流程图。

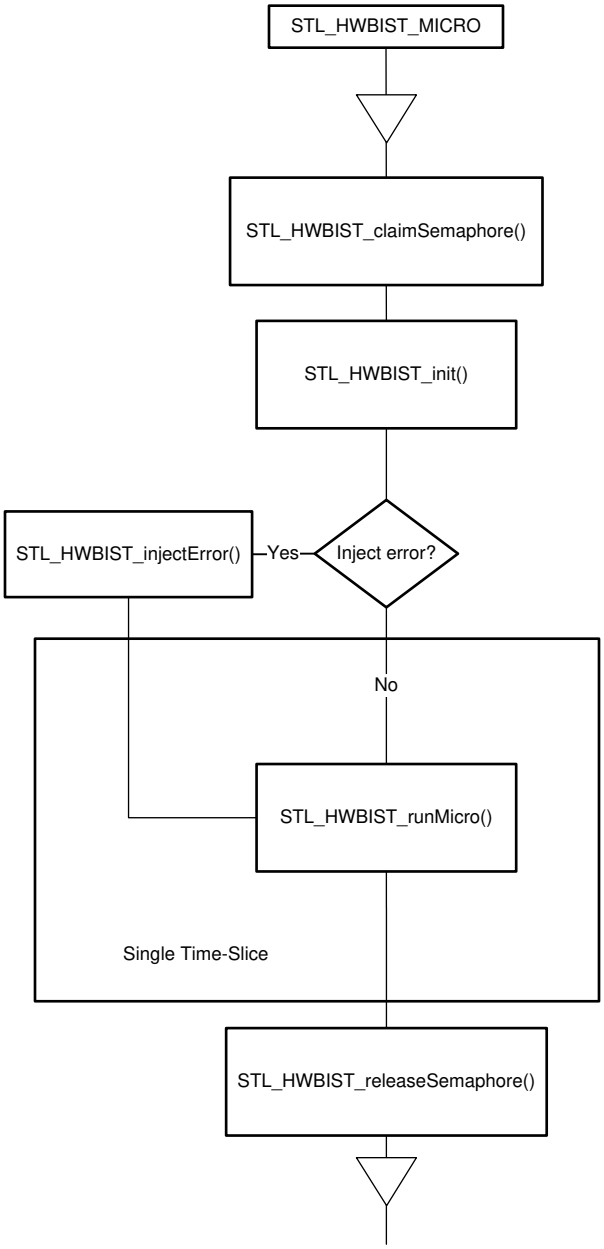

图 2-2. 时间分片微运行执行的流程图

2.1.2.2 执行 HWBIST 完全运行

要执行待测 CPU 的 HWBIST 的完整运行，请调用以下函数：

```
STL_HWBIST_runFull();
```

errorType 参数为枚举类型 STL_HWBIST_Error，它指定在执行 HWBIST 测试的完整运行之前要注入的错误类型。如果在多核器件上，此函数要求 CPU 先尝试运行完整的 HWBIST 以声明 HWBIST 信标，然后再调用该函数。此函数初始化 HWBIST 引擎，然后注入 *errorType*。它还将 STL_HWBIST_errorNMIISR() 函数注册为 NMI 处理程序。然后，该函数执行完整的 HWBIST 运行，从而实现器件支持的最大覆盖率。如果 HWBIST 中存在故障，则设置全局错误标志，而返回值指定故障。此外，如果在预期的微运行中没有实现覆盖率，则测试会因超限而失败。在返回之前，该函数恢复之前的 NMI 矢量。

如果 HWBIST 完整运行测试合格，在预期的微运行次数内没有出现错误，则此函数返回 HWBIST 的状态，该值将为值 *STL_HWBIST_BIST_DONE* 和 *STL_HWBIST_MACRO_DONE* 的按位或结果。如果测试失败，则 HWBIST 的状态和函数的返回值是以下各值的某种组合的按位或结果：*STL_HWBIST_NMI*、*STL_HWBIST_BIST_FAIL*、*STL_HWBIST_INT_COMP_FAIL*、*STL_HWBIST_TO_FAIL* 和 *STL_HWBIST_OVERRUN_FAIL*。

有关这些类型的错误的更多信息，请参阅《SDL 用户指南》。

图 2-3 展示了一个流程图，该图详细介绍了器件上支持 99% 覆盖率的 STL_HWBIST_runFull() 函数的设计。

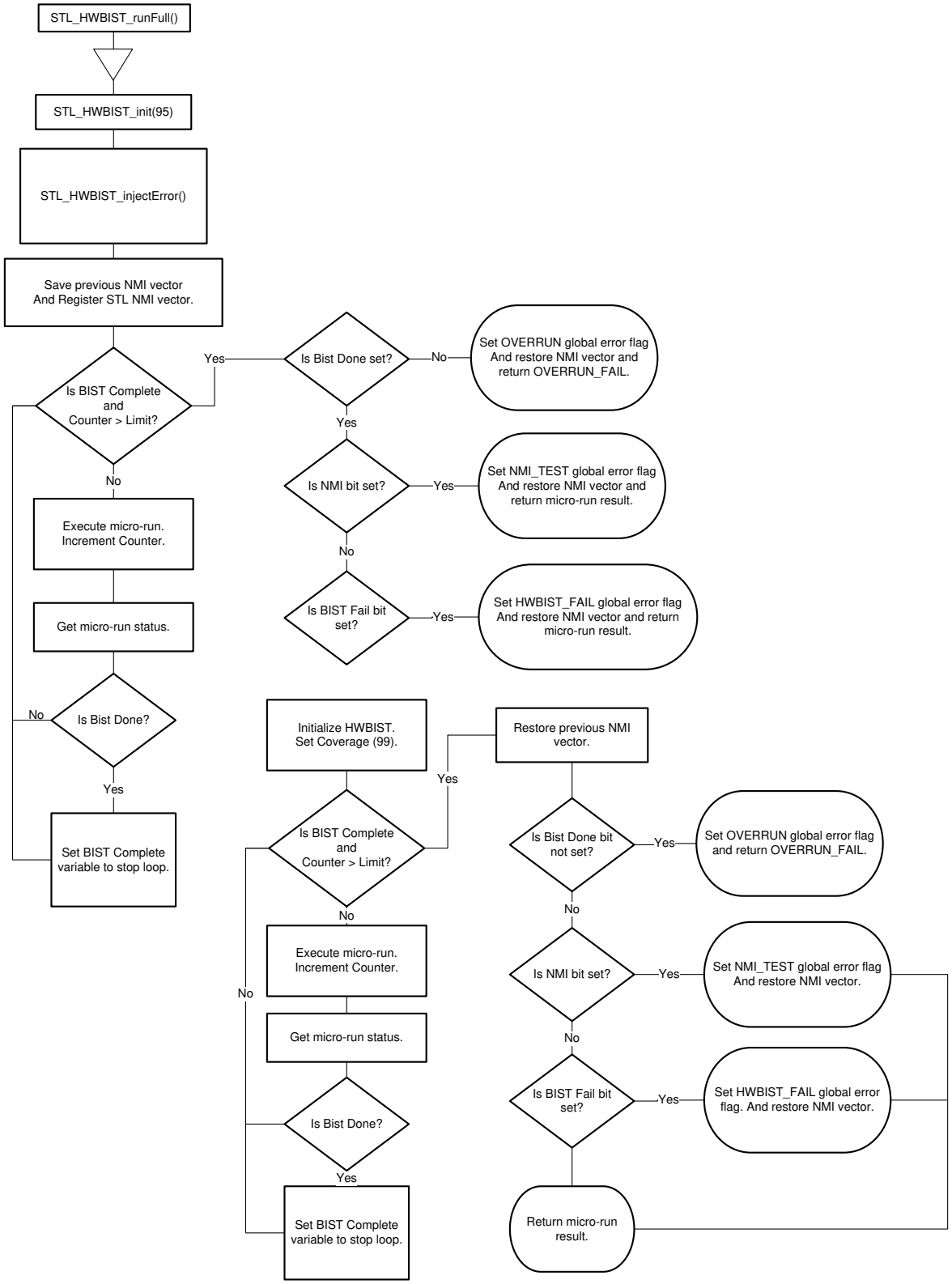

图 2-3. STL_HWBIST_runFull() 流程图

要使用 STL_HWBIST_runFull() 函数执行覆盖率为 99% 的完整 HWBIST，请执行以下函数序列：

1. 如果在多核器件上，请声明 HWBIST 信标。

```
STL_HWBIST_claimSemaphore();
```

2. 执行 HWBIST 完全运行。

```
STL_HWBIST_runFull();
```

3. 如果在多核器件上，请释放 HWBIST 信标。

```
STL_HWBIST_releaseSemaphore();
```

该序列在单个时间片中执行完整的 HWBIST。图 2-4 展示了单个时间片中的完整 HWBIST 运行。

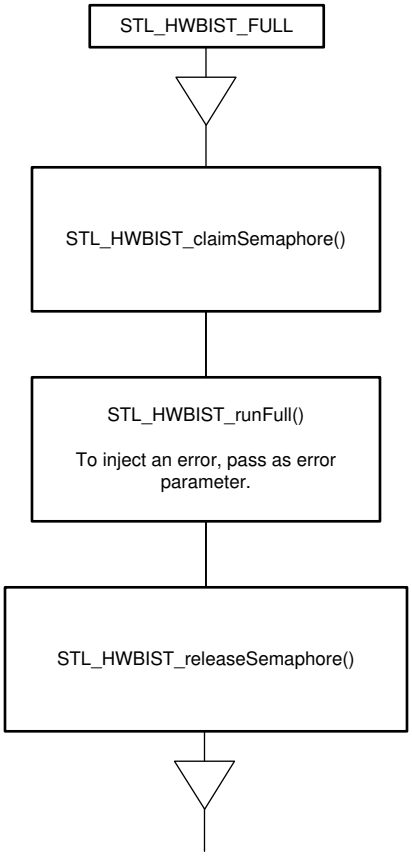

图 2-4. 单个时间片中的完整 HWBIST

2.1.3 错误管理

HWBIST 执行过程中出现故障是一种严重的情况。如果发生这种情况，则无法确保发生故障的 CPU 的行为。必须包含采取适当措施以正常关闭系统的代码。这可以通过解码 STL_HWBIST_runFull() 或 STL_HWBIST_runMicro() 的返回值来实现。

可通过 NMI 的陷阱管理代码。这是一种管理 HWBIST 故障的更快方法，尤其是在双 CPU 器件的情况下，因为 NMI 同时发送到两个 CPU。要利用 NMI 陷阱，系统代码必须执行以下操作：

1. 清除 NMI 标志寄存器的任何 NMI 陷阱残留：

```
SysCtl_clearNMIStatus(STL_HWBIST_NMI_CPU1_HWBISTERR);

SysCtl_clearNMIStatus(STL_HWBIST_NMI_CPU2_HWBISTERR);
```

2. 将 NMI 矢量映射到处理 HWBIST 的中断服务例程：

```
Interrupt_register(INT_NMI, STL_HWBIST_errorNMIISR);
```

3. 启用 PIE 控制器：

```
Interrupt_enablePIE();
```

   PIECTRL 寄存器的 ENPIE 位在 HWBIST 软件正在测试的 CPU 上设置，但也必须在另一个 CPU（在双 CPU 器件中）上设置该位以响应故障。

有关 NMIFLG 寄存器及其包含的 HWBIST 标志的更多详细信息，请参阅特定于器件的《技术参考手册》。

2.2 管理双核器件上的 HWBIST

F2837xD 和 F2838xD 双核器件支持对每个内核进行 HWBIST 测试。一次只能有一个内核运行 HWBIST。为了管理此 HWBIST 控制器，某些信标寄存器允许一个处理器拥有 HWBIST 控制器，直到其完成运行。完成完整 HWBIST 后，待测处理器应将信标控制权释放给另一个处理器，以便它可以运行 HWBIST。

2.2.1 信标管理

使用以下函数调用处理信标管理：

```
bool STL_HWBIST_claimSemaphore(const STL_HWBIST_Core core);

void STL_HWBIST_releaseSemaphore(void);
```

- 系统复位后，HWBIST 信标 = 0。
  - CPU1 可以访问 HWBIST 资源并更改信标。
  - CPU2 可以更改信标。
- 当 CPU1 决定执行 HWBIST 时，它设置信标 = 2。
  - 授予 CPU1 对 HWBIST 资源的访问权限
  - 阻止 CPU2 访问 HWBIST 资源并阻止更改信标
- 当 CPU1 完成 HWBIST 测试时，它设置信标 = 3，这授予任一 CPU 访问信标的权限。
- 当 CPU2 决定执行 HWBIST 时，它设置信标 = 1。
  - 授予 CPU2 对 HWBIST 资源的访问权限。
  - 阻止 CPU1 访问 HWBIST 资源并阻止更改信标。
- 当 CPU2 完成 HWBIST 测试时，它设置信标 = 3，这授予任一 CPU 访问信标的权限。

2.2.2 处理器间通信

处理器间通信 (IPC) 外设可以轻松管理所需的通信级别，以便在其中一个处理器（CPU1 和 CPU2）打算运行 HWBIST 时通知每个处理器。例如，如果必须监控关键系统中断并将其映射到 CPU1，则在 CPU1 执行 HWBIST

操作时将此中断映射到 CPU2 可能是有利的。可以利用 IPC 消息和中断来实现 CPU1 和 CPU2 之间的这种处理器间通信。在处理信标管理时，IPC 还可用于在两个处理器之间实现一定程度的握手。

2.3 使用 HWBIST 时的系统注意事项

总之，当 HWBIST 微运行执行时，目标 CPU 出于所有实际目的从系统中消失了。

2.3.1 中断延迟

运行极小规模的微运行所需的时间长度因器件系列而异，因此，您应该参考《SDL 用户指南》中提供的 STL_HWBIST_runMicro() 函数的分析数据。一般来说，此过程在 200MHz SYSCLK 器件上大约需要 2.5µs，而在 100MHz SYSCLK 器件上大约需要 4µs。这些值是使用从 0 等待状态 SRAM 运行的 HWBIST 函数测量的，如果使用闪存的等待状态，则需要的时间更长。这需要考虑以下几点：

- *是否有任何系统关键中断无法等待此延迟？*

  考虑由于已识别的系统故障而想要关闭控制操作的中断。如果有，则必须将这些中断重新路由到另一个处理器或 DMA 通道以进行紧急处理，而 HWBIST 微运行拥有 CPU 电路。此外，系统关键中断或任务可以映射到 NMI，这将停止 HWBIST 微运行执行。
- *控制环路是否需要在此延迟期内根据反馈进行更新？*

  如果是这样，请勿启动 HWBIST，直到此更新完成，并且在下一个更新之前，您有足够的时间；或者使用 DMA 通道管理更新。
- *控制环路中是否有时间片可用于执行 HWBIST 微运行？*

  如果有，则这可能是执行 HWBIST 微运行的适用时隙。

2.3.2 电源注意事项

极小规模的微运行比在 CPU 上运行的代码消耗的功率更多。您应该考虑以下几点：

- 在上电复位 (POR) 和引导 ROM 启动代码完成后，大多数外设并未立即运行。因此，如果在系统开始执行控制环路之前运行 HWBIST，则可以有足够的功率裕度来处理额外的电流。
- 如果在大量器件电路处于活动状态且处于高温状态时执行 HWBIST，请采取以下任一措施：
  - 在系统设计中包括一些额外的功率裕度
  - 使用 /2 或 /4 时钟执行微运行

第二个选项将执行和中断延迟时间分别增加至 2 倍或 4 倍。此外，此选项要求更改源代码，并对软件诊断库中的 HWBIST 函数进行额外的测试。要将时钟除以 2，必须将值 1 写入 CSTCGCR7 的位 18-19。要将时钟除以 4，必须将值 2 写入 CSTCGCR7 的位 18-19。应在 stl_hwbist.c 中 STL_HWBIST_init() 函数的适当位置进行此源代码修改。

2.3.3 HWBIST 存储器要求

为 HWBIST 运行保留了三个存储器范围，如下所示：

- 从 CPU 地址 0x0000 开始的 32 个字。这是 HWBIST 复位后引导 ROM 跳转到的入口点，由 *hwbist* 存储器段使用。它无法移动到另一个地址。
- 用于保存和恢复完整环境的 *hwbiststack* 段。它必须位于 SP 可寻址区域内。保存/恢复的大小因器件系列而异。与需要容纳 FPU64 寄存器的 F2838x 一样，至多需要 82 个字。
- 许多 HWBIST 函数都放在 *.TI.ramfunc* 中以获得更佳性能。大小可能会略有不同，具体取决于器件系列以及编译器版本和选项。您可以从 HWBIST 库文件中删除 CODE_SECTION #pragma，如果希望从闪存中执行它们，则可以重新构建库。

2.3.4 注入错误

HWBIST 包括一些错误注入功能，以帮助验证系统错误处理代码。可以通过运行以下函数来调用这些错误：

```
void STL_HWBIST_injectError(const STL_HWBIST_Error errorType);
```

表 2-1 列出了注入的错误类型、值和预期行为。

表 2-1. 注入错误、值和行为

STL_HWBIST_ERROR 类型	值	描述和行为
STL_HWBIST_NO_ERROR	0x00000000	为将来的操作清除错误注入特性。 如果不存在故障，HWBIST 在正常操作下获得通过。
STL_HWBIST_TIMEOUT	0x0000000A	调用超时错误 这与 HWBIST 控制器中的计时器相关。如果 HWBIST 控制器在微运行期间超时，则微运行已失去控制。 这将生成一个超时故障标志，并向该 CPU（如果是双核器件，则向这两个 CPU）生成 NMI。
STL_HWBIST_FINAL_COMPARE	0x000000A0	损坏 MISR 比较 HWBIST 正常执行，但在完成时与损坏的 MISR 进行比较。这不会导致故障状况，但它确实允许 CPU 检查 MISR 是否存在电路问题。 不会生成 NMI，也不会生成故障状态。
STL_HWBIST_NMI_TRAP	0x00000A00	强制向 HWBIST 控制器生成一个 NMI 以调用关断，并将控制权返回给 CPU。 在开始 HWBIST 微运行执行之前停止微运行，并向待测 CPU 生成 NMI。 **备注** 尽管触发了 NMI，但没有设置 NMI 标志。
STL_HWBIST_LOGIC_FAULT	0x00002000	将逻辑错误注入待测电路，以查看 HWBIST 是否捕获到该错误 这会导致设置相应的 HWBIST 故障状态位，并向该 CPU（如果是双 CPU 器件，则向这两个 CPU）生成 NMI。 **备注** 有效的逻辑错误注入值为 0x00001000 到 0xFFFFF000。诊断库仅提供一个值 (0x00002000)。但是，您可能希望修改源代码，以允许将其他或多个逻辑错误注入值写入 CSTCTEST 寄存器。请参阅软件诊断库的 stl_hwbist.h 中的源代码。

如果在调试错误管理过程中代码丢失，极可能的原因是 NMI 执行未适当地进行初始化。有关详细信息，请参阅节 2.1.3。

2.4 调试系统内 HWBIST

在执行 HWBIST 微运行时，出于所有实际目的，与目标 CPU 的仿真连接将从系统中断开。这意味着，诸如断点、观察点、单步执行甚至运行之类的功能不可用于调试系统代码。这来自 HWBIST 运行的以下两个方面：

- 与 CPU 一样，扫描仿真分析电路的过程正在执行，因此断点（和其他仿真分析功能）由 HWBIST 控制器主动损坏的锁存器进行管理。
- 无法保存和恢复分析电路的环境。

TI 建议您在执行 HWBIST 时不要在代码中启用软件断点。因此，在调试系统代码时需要禁用 HWBIST 操作。在验证或调试 HWBIST 操作时，必须使用 Free Run 操作启动 CPU 代码执行。如果运行双核器件，则必须使用 Free Run 操作启动这两个 CPU。在 CPU 上运行 HWBIST 时，建议删除与该特定 CPU 的交叉触发器（如果已设置）的任何关联。

以下是一些有关调试 HWBIST 代码的帮助提示：

- 在执行期间，使用电路板上的可观察点来监控进度。例如，使用一个或多个连接到示波器或连接到 LED 的 GPIO 引脚。
- 您可以在代码中放置循环以代替断点来“停止”它，并跟踪其进度。
- 将调试和进度更新或状态存储在 SRAM 中：
  - 理想情况下，这是在 BootROM 或仿真器 GEL 脚本未初始化的存储器的范围内。
  - 如果在双核器件上，尽可能将这些更新存储在两个 CPU 都可以访问的存储器中：
    - 例如，共享存储器、IPC 寄存器或消息 RAM。
    - 未运行 HWBIST 的 CPU 可能能够完全停止，并显示调试和进度信息。
- 有时，在运行 HWBIST 之后，仿真器停止操作会调用以下弹出消息：

  停止目标 CPU 时出现问题：（错误 -1156 @ 0x0）。器件可能在低功耗模式下运行。是否要使其退出此模式？（仿真包 5.1.636.0）。

  此消息是正常的，是因仿真器与处理器失去同步引起的。仿真器始终失去同步，但有时会在没有此操作的帮助下重新获得同步。这与低功耗模式无关。

  - 如果发生这种情况，请点击“Yes”按钮。
  - 如果这不起作用，则可以断开 CPU 并重新连接以重新获得控制权。在双核器件上，也许仍然可以访问其他 CPU。因此，如果已保存调试和进度值，则另一个 CPU 可以提供对它们的访问。
- 如果 CPU 恢复正常，但它会引导至 BootROM 或闪存，一个可能的原因是未启用 PIE。HWBIST 在完成后执行 CPU 复位，但如果未启用 PIE，则 CPU 会引导至 BootROM，而不是诊断库 STL_HWBIST_restoreContext() 代码。

3 参考文献

- 德州仪器 (TI)：*TMS320F2837xD 双核微控制器技术参考手册*
- 德州仪器 (TI)：*TMS320F2837xD 双核微控制器数据表*
- 德州仪器 (TI) ：*TMS320F2837xD 双核微控制器器件勘误表*
- 德州仪器 (TI) ：*TMS320F2837xS 微控制器技术参考手册*
- 德州仪器 (TI) ：*TMS320F2837xS 微控制器数据表*
- 德州仪器 (TI) ：*TMS320F2837xS 微控制器器件勘误表*
- 德州仪器 (TI) ：*TMS320F2807x 微控制器技术参考手册*
- 德州仪器 (TI) ：*TMS320F2807x 微控制器数据表*
- 德州仪器 (TI) ：*TMS320F2807x 微控制器器件勘误表*
- 德州仪器 (TI)：*TMS320F2837xD、TMS320F2837xS 和 TMS320F2807x 的功能安全手册*
- 德州仪器 (TI)：*具有连接管理器的 TMS320F2838x 实时微控制器技术参考手册*
- 德州仪器 (TI)：*具有连接管理器的 TMS320F2838x 实时微控制器数据表*
- 德州仪器 (TI)：*TMS320F2838x 实时微控制器器件勘误表*
- 德州仪器 (TI)：*TMS320F28003x 实时微控制器技术参考手册*
- 德州仪器 (TI)：*TMS320F28003x 实时微控制器数据表*
- 德州仪器 (TI)：*TMS320F28003x 实时微控制器器件勘误表*
- 德州仪器 (TI)：*TMS320F28002x 实时微控制器技术参考手册*
- 德州仪器 (TI)：*《TMS320F28002x 实时微控制器》数据表*
- 德州仪器 (TI)：*TMS320F28002x 实时微控制器器件勘误表*
- 德州仪器 (TI)：*TMS320F28002x 安全手册*
- 德州仪器 (TI)：*C2000™ 实时微控制器的汽车功能安全特性*
- 德州仪器 (TI)：*C2000™ 实时微控制器的工业功能安全特性*
- 适用于 F2837xD、F2837xS 和 F2807x 器件的 C2000™ SafeTI™ 诊断软件库
- C2000Ware，包含适用于 F2838x、F28003x 和 F28002x 的软件诊断库

4 Revision History

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (October 2017) to Revision A (September 2022)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式。	2
• 更新了 节 2.4。	18
• 更新了 节 3	19

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265