查看所有版本

Vivado Design Suite 用户指南

烧录和调试

UG908 (v2024.2) 2024 年 11 月 13 日

目录

第 1 章

简介

按设计进程浏览内容

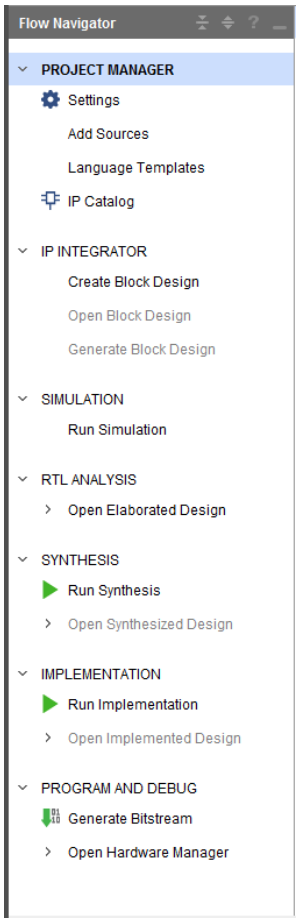
AMD 自适应计算文档按一组标准设计进程进行组织，以便帮助您查找当前开发任务相关的内容。您可以在设计中心页面上访问 AMD Versal™ 自适应 SoC 设计进程。您还可以使用设计流程助手来更深入了解设计流程，并找到特定于预期设计需求的内容。本文档涵盖了以下设计进程：

- 硬件、IP 和平台开发：为硬件平台创建 PL IP 块、创建 PL 内核、功能仿真以及评估 AMD Vivado™ 时序收敛、资源使用情况和功耗收敛。还涉及为系统集成开发硬件平台。本文档中适用于此设计进程的主题包括：
  - 第 11 章：设计调试
  - 第 12 章：系统内逻辑设计调试流程
  - 第 13 章：在硬件中调试逻辑设计
  - 第 14 章：在波形查看器中查看 ILA 探针数据
  - 第 15 章：实现后的设计调试
- 开发板系统设计：通过板级原理图和开发板布局来设计 PCB。还包含功耗、散热以及信号完整性注意事项。本文档中适用于此设计进程的主题包括：
  - 第 4 章：器件烧录
  - 第 6 章：在 Vivado 中执行远程调试
  - 第 7 章：配置存储器器件烧录
  - 第 8 章：高级烧录功能
  - 第 10 章：串行矢量格式 (SVF) 文件烧录
  - 第 16 章：串行 I/O 硬件调试流程
  - 第 18 章：在硬件中调试串行 I/O 设计
  - 附录 A: 器件配置比特流或 PDI 设置
  - 附录 C: 低级别 SVF JTAG 命令
  - 附录 D: hw_server 支持的 JTAG 线缆和器件
  - 附录 F: 配置存储器支持

入门指南

成功实现设计后，下一步是在硬件中运行该设计，具体方式是对 FPGA 或自适应 SoC 进行烧录并进行系统内设计调试。执行 FPGA 烧录和设计的系统内调试所需的所有必要命令都包含在 AMD Vivado™ 集成设计环境 (IDE) 的 Flow Navigator 的“Program and Debug”（烧录和调试）部分中。

图 1：**Flow Navigator 面板的“Program and Debug”部分**



调试术语

ILA

Integrated Logic Analyzer (ILA) 功能支持您在 FPGA、SoC 或 AMD Versal™ 器件上对实现后的设计执行统内调试。需要监控设计内的信号时，应使用此功能。另外，您还可以使用此功能触发硬件事件并以系统级速度捕获数据。

ILA 核心可在 RTL 代码中例化，或者也可在 Vivado 设计流程中完成综合后插入。如需获取有关 ILA IP 核的详细文档记录，请参阅《Integrated Logic Analyzer LogiCORE IP 产品指南》(PG172)。

相关信息

系统内逻辑设计调试流程
在硬件中调试逻辑设计

VIO

Virtual Input/Output (VIO) 调试功能可实时监控和驱动内部 FPGA、SoC 或 Versal 自适应 SoC 信号。如果无法通过物理方式访问目标硬件，则可使用此调试功能来驱动并监控真实硬件上存在的信号。

此调试核需在 RTL 代码中例化，因此您需要事先明确要驱动的信号线。该核列在 IP 目录的“Debug”（调试）类别下。如需获取有关 VIO IP 核的详细文档记录，请参阅《Virtual Input/Output LogiCORE IP 产品指南》(PG159)。

相关信息

在硬件中调试逻辑设计

IBERT

Integrated Bit Error Ratio Tester (IBERT) Serial Analyzer 设计支持系统内串行 I/O 确认和调试。这样您即可在基于 FPGA 的系统内对自己的高速串行 I/O 链路进行测量和优化。AMD 建议使用 IBERT Serial Analyzer 设计来解决各种系统内调试和确认问题，从简单的时钟设置和连接问题到复杂的裕度分析和通道优化问题都不在话下。

AMD 建议在对接收到的信号应用接收器均衡后，使用 IBERT Serial Analyzer 设计来测量信号质量。这样可确保在发射到接收通道中的最优点执行测量，从而确保获取真实准确的数据。用户可通过在 IP 目录中选择、配置和生成 IBERT 核并选择该核的“Open Example Design”（打开设计示例）功能来访问此设计。请参阅 串行 I/O 硬件调试流程 和 第 18 章：在硬件中调试串行 I/O 设计 以获取有关 IBERT 核以及在 Vivado Design Suite 中的使用方法论的更多详细信息。

相关信息

在硬件中调试串行 I/O 设计
串行 I/O 硬件调试流程

JTAG-to-AXI Master

注释：在 Versal 自适应 SoC 器件上不支持 JTAG-to-AXI Master，因为内置 CIPS AXI Master 接口可搭配 Debug Packet Controller (DPC) 来生成 AXI 传输事务，而无需其他 IP。

JTAG-to-AXI Master 调试功能用于生成 AXI 传输事务，这些传输事务将与硬件中运行的系统中的各种 AXI4 和 AXI4-Lite 从核进行交互。AMD 建议您在运行时，使用该核在 FPGA 内部生成 AXI 传输事务以及调试或驱动 AXI 信号。该核也可在无处理器的设计内使用。

该核列在 IP 目录的“Debug”（调试）类别下。本指南的“在硬件中调试逻辑设计”部分包含有关 JTAG-to-AXI Master 核及其在 Vivado Design Suite 中的使用方法论的详细信息。如需获取有关 JTAG-to-AXI IP 核的详细文档记录，请参阅《JTAG to AXI Master LogiCORE IP 产品指南》(PG174)。

相关信息

在硬件中调试逻辑设计

Debug Hub

在 7 系列和 AMD UltraScale™ 架构上，Vivado Debug Hub 核可在 FPGA 的 JTAG 边界扫描 (BSCAN) 接口与下列类型的 Vivado 调试核之间提供 1 个接口：

- Integrated Logic Analyzer (ILA)
- Virtual Input/Output (VIO)
- Integrated Bit Error Ratio Tester (IBERT)
- JTAG-to-AXI
- 存储器 IP

> **重要提示！** Vivado Debug Hub 核无法例化到设计中。该核由 Vivado 在 `opt_design` 阶段中插入。

AXI4 Debug Hub

在 AMD Versal™ 自适应 SoC 架构上，AXI4 Debug Hub 作为 IP 核，可在 CIPS 的 AXI4 主接口与 Vivado 硬件调试核上的 AXI4-Stream 接口之间提供接口，其中包括：

- Integrated Logic Analyzer (ILA)
- Virtual Input/Output (VIO)
- 软核存储器 IP

注释： 像先前架构一样，在 Versal 器件上，AXI4 Debug Hub 既可手动例化为 IP，也可在执行 `opt_design` 期间自动插入。

System ILA

System Integrated Logic Analyzer (System ILA) IP 核是一种逻辑分析仪，它支持您对 FPGA 器件上的实现后的设计执行系统内调试。如需监控 IP integrator 块设计中的接口和信号，请使用此 IP。另外，您还可以使用此功能来触发硬件事件相关的接口和信号并以系统级速度捕获数据。这样可确保对 FPGA 或自适应 SoC 上的设计进行调试时，能够在硬件管理器中直观演示接口事件。此 IP 可提供 AXI 接口调试和监控功能以及 AXI4-MM 和 AXI4-Stream 协议检查能力。

由于 System ILA 核与受监控的设计同步，因此应用于您的设计的所有设计时钟约束也同样会应用于该 System ILA 核的组件。如需获取有关 System ILA IP 核的详细文档记录，请参阅《System Integrated Logic Analyzer LogiCORE IP 产品指南》(PG261)。

注释： 在 AMD Versal™ 器件上，可通过使用 Versal ILA 核来使用 System ILA。

Debug Bridge

注释： 在 Versal 架构上不支持 Debug Bridge IP。

Debug Bridge IP 核属于可提供多个选项的控制器，用于与设计中的调试核进行通信。

Debug Bridge 的主要用例是使用 AMD 虚拟线缆 (XVC) 通过以太网或其他接口远程调试设计，而无需使用 JTAG 线缆。

另一种常见用例是用于调试 Dynamic Function eXchange 和 Tandem PCIe with Field Updates（含现场更新的串联 PCIe）设计。如需了解有关 Tandem PCIe with Field Updates 流程和 Debug Bridge 的更多信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

在 JTAG 并非首选通信和调试机制的系统中，也可将 Debug Bridge 与 PCIe® 核搭配使用。如需了解有关将 PCIe 核与 Debug Bridge 搭配使用的 XVC 流程的更多信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

如需获取有关 Debug Bridge IP 核的详细文档，请参阅《Debug Bridge LogiCORE IP 产品指南》(PG245)。

In-System IBERT

注释：In-System IBERT 仅在 UltraScale 和 UltraScale+ 上受支持。

In-System IBERT IP 支持您使用 Vivado Serial I/O Analyzer 对设计中的 UltraScale 和 UltraScale+ 收发器执行 2D 眼图扫描。当收发器与系统其余部分进行交互时，此 IP 会使用来自设计的数据来实时绘制收发器眼图。此 IP 可与设计中的用户逻辑或基于 AMD 收发器的 IP（例如，GT Wizard 或 Aurora）集成。

如需获取有关 In-System IBERT IP 的详细文档记录，请参阅《In-System IBERT LogiCORE IP 产品指南》(PG246)。

IBERT GTR

IBERT UltraScale+ GTR 可用于评估和监控 Zynq UltraScale+ MPSoC 中的 GTR 收发器。通过此功能，您可以完成以下任务：

- 利用用户数据执行眼图扫描
- 更改 GTR 设置
- 查看链路状态
- 检查所有 GTR 通道使用的所有 PLL 的“锁定”状态

但 IBERT GTR 无法提供以下功能：

- 利用原始 PRBS 数据模式执行眼图扫描
- 测量误码率（无比特计数器或误差计数器）

注释：这是基于软件的解决方案，即，在器件的可编程逻辑中无需 IP 或逻辑。

第 2 章

Vivado Lab Edition

AMD Vivado™ Lab Edition 是完整版 Vivado Design Suite 的独立安装版本，包含在生成器件镜像后对 AMD 器件进行烧录和调试所需的所有功能。通常适用于在如下实验室环境内进行烧录和调试：实验室环境中的机器所含磁盘空间、内存和连接资源较少。Vivado Lab Edition 占用资源较少，安装包大小为 1 GB，安装后占用空间约 2.4 GB。

安装

要安装 Vivado Lab Edition，请从 Unified Installer 中选择 Lab Edition。

如需获取详细的安装、许可与版本信息，请参阅《Vivado Design Suite 用户指南：版本说明、安装和许可》(UG973)。

在 Windows 上启动 Vivado Lab Edition

要启动 Vivado Lab Edition，请依次单击：

“Start” → “All Programs” → “Xilinx Design Tools” → “Vivado Lab 2024.2”

在 Windows 或 Linux 上从命令行启动 Vivado Lab Edition

在命令提示符处输入以下命令：

```
vivado_lab
```

提示：要在命令提示符处运行 `vivado_lab`，请使用以下脚本来设置您的环境：

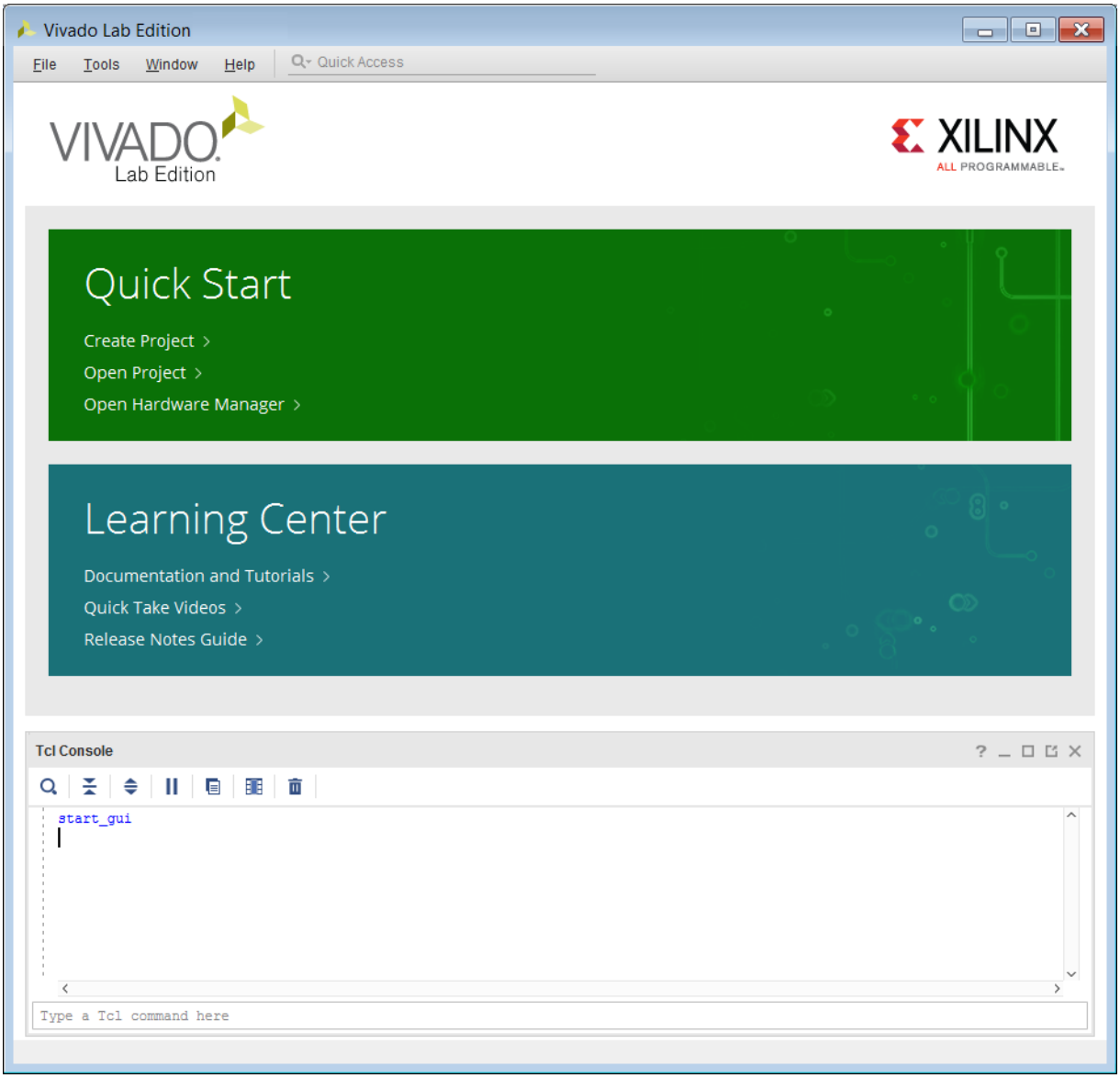
```
C:\Xilinx\Vivado_Lab\2023.x\settings64.(bat|sh)
```

您可从任意目录打开 Vivado Lab Edition。但 AMD 建议从可写入的工程目录运行 Vivado Lab Edition，因为其 log 日志文件和 journal 日志文件将写入启动目录。从命令提示符运行时，请从工程目录启动 Vivado IDE，或者使用 `vivado_lab -log` 和 `journal` 选项来指定位置。使用 Windows 快捷方式时，必须修改快捷方式的“Start in folder”（从文件夹启动）属性。如果不从可写入的工程目录启动，则会导致出现警告，并且该工具可能出现不可预测的行为。

使用 Vivado Lab Edition

启动 Vivado Lab Edition 时，会显示“Getting Started”（入门指南）页面（请参阅下图）并为您提供不同选项，以帮助您开始使用 Vivado Lab Edition。

图 2：**Vivado Lab Edition 欢迎屏幕**



开始处理工程

要对设计进行烧录或调试，可以创建或打开工程，然后连接到目标服务器和器件。“Getting Started”（入门指南）页面的“Quick Start”（快速入门）部分提供了便于访问以下任务的相应链接：

- 创建工程。
- 打开现有工程

  注释：您也可以从“Recent Projects”（最近的工程）列表中打开最近访问的工程。

打开硬件管理器

您可打开 AMD Vivado™ Design Suite 硬件管理器以将自己的设计比特流下载至器件。硬件管理器的 Vivado Logic Analyzer 和 Vivado Serial I/O Analyzer 功能可用于调试设计。例如，您可将 ILA、VIO 和 JTAG-to-AXI 核添加到自己的设计中，以便在 Vivado Logic Analyzer 中进行调试，或者也可以使用来自 AMD IP 目录的 IBERT 设计示例，通过 Vivado Serial I/O Analyzer 对设计中的 GT 进行测试和配置。

复查文档和视频

在“Getting Started”（入门指南）页面上，您可使用 AMD Documentation Navigator 来访问各种文档，包括用户指南、教程、视频和版本说明等。

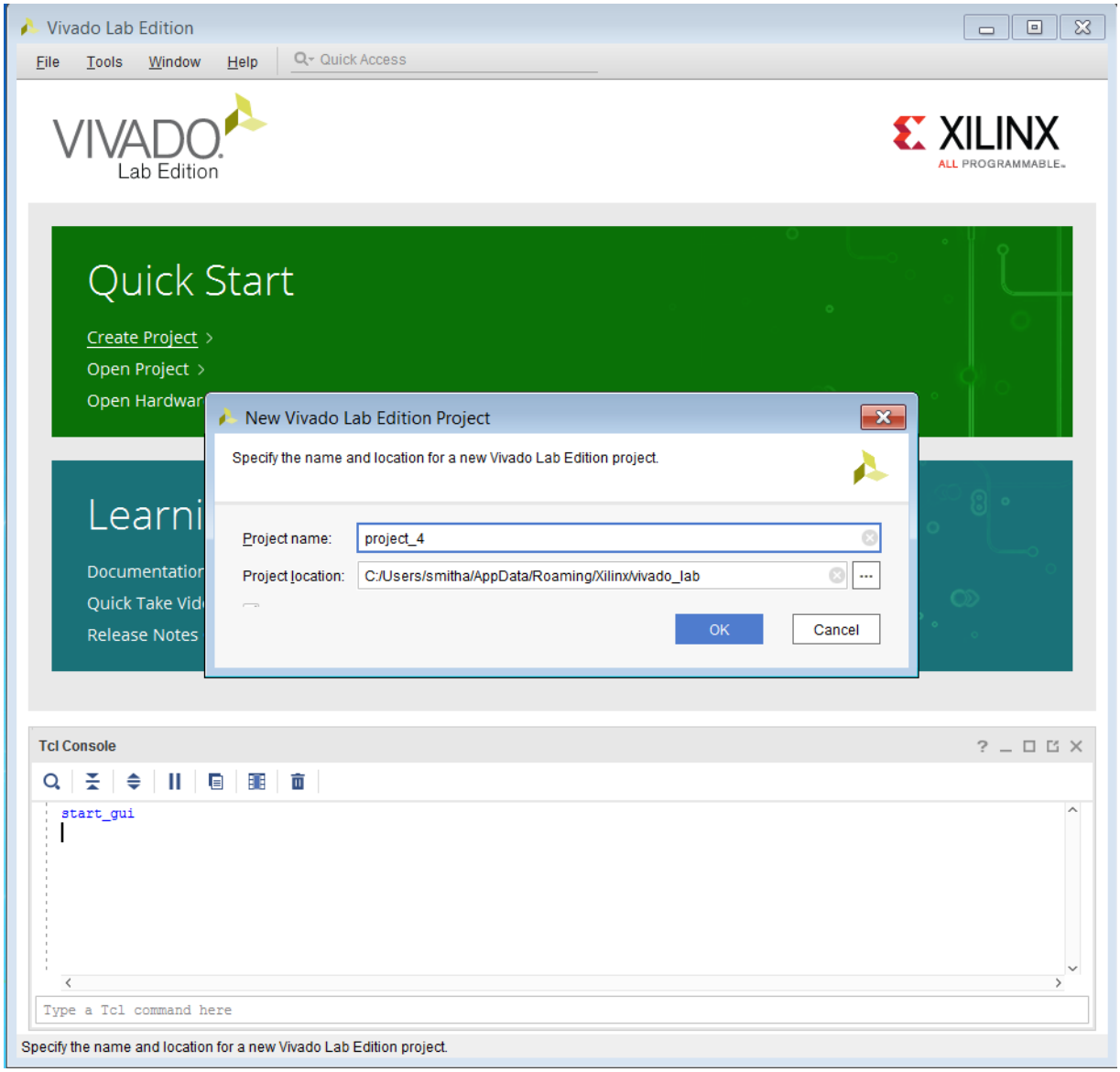
Vivado Lab Edition 工程

Vivado Lab Edition 允许用户在实验室内创建工程。所有相关烧录和运行时调试首选项和设置都存储在该工程内。重新打开该工程时，这些设置和首选项将复原到该工具中。在 Vivado Lab Edition 工具和 Vivado Design Suite 中均可创建 Vivado Lab Edition 工程。

创建新工程

要在 Vivado Lab Edition 中创建新工程，请单击“Create New Project”（创建新工程）图标，如下图所示。在“New Vivado Lab Edition Project”（新建 Vivado Lab Edition 工程）对话框中输入工程名称和位置。创建新工程时，Vivado Lab Edition 会创建工程文件。此工程文件名与“New Vivado Lab Edition Project”对话框中输入的工程名称相同，且带有 `.lpr` 扩展名。请参阅下图。

图 3：**Vivado Lab Edition 创建新工程**



使用 Tcl 命令创建工程

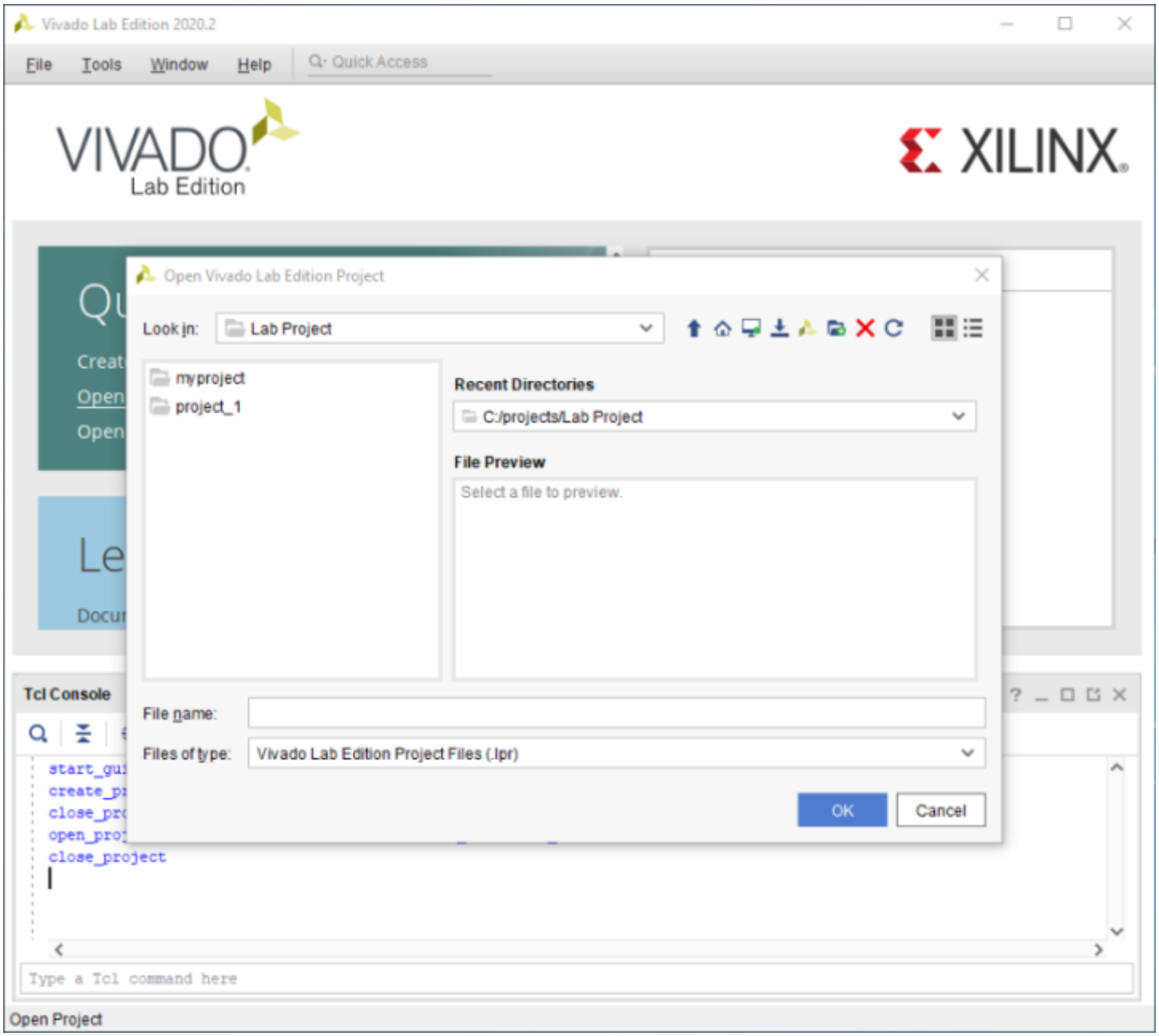
您也可使用 Tcl 命令创建工程。在 Vivado Lab Edition 的 Tcl 控制台中输入以下命令，或者使用 source 命令从 Tcl 文件中找到并运行这些命令。

```
create_project project_1 C:/Lab_edition/project_1
```

打开工程

要打开现有工程，请单击打开工程图标（如下图所示）或者双击“Recent Projects”（最近的工程）列表中的工程。这样即可打开浏览器，以便您打开任意 Vivado Lab Edition 工程文件（扩展名为 `.lpr`）。默认情况下，在“Recent Projects”列表中会列出最近打开的 10 个工程。要更改此数量，请单击“Tools” → “Settings”（工具 > 设置）并更新“Project”（工程）选项。Vivado Lab Edition 会先检查确认工程数据可用，然后再显示工程。

图 4：**Vivado Lab Edition 打开工程对话框**



使用 Tcl 命令打开工程

您还可使用 Tcl 命令打开工程。在 Vivado Lab Edition 的“Tcl Console”（Tcl 控制台）中输入以下命令，或者使用 source 命令从 `.tcl` 文件中找到并运行这些命令。

```
open_project C:/Lab_edition/project_1/project_1.lpr
```

在 Vivado Lab Edition 中使用现有器件镜像和调试探针文件

您可使用现有器件镜像（`.bit` 或 `.pdi`）和 `.ltx` 文件，此镜像和文件源自装有 Vivado Lab Edition 的实验室机器中先前运行的实现。

典型流程包括：

1. 创建新的 Vivado Lab Edition 工程。
2. 连接到开发板。
3. 为工程指定 `.bit` 或 `.pdi` 文件和 `.ltx` 文件。
4. 您可将这些文件手动复制到网络驱动器，或者将其直接指向网络驱动器。
5. 烧录器件。
6. 在硬件中调试设计。
7. 更改结果将即时保存到工程中。
8. 用户首选项、运行时管理器调试仪表板和窗口设置将即时保存到工程中。
9. 重新打开工程时，用户首选项、运行时管理器调试仪表板和窗口设置都将复原。

使用来自 Vivado Design Suite Edition 的现有 .lpr 工程

当您使用硬件管理器对工程中的设计进行烧录和/或调试时，Vivado Design Suite 会在工程启动时创建 `.lpr` 文件，并在其中填充相应的详细信息。此文件位于 `project_name.hw` 目录中，且名为 `project_name.lpr`。在 Vivado Lab Edition 中可打开此工程文件。

典型流程包括：

1. 单击 Vivado Lab Edition 起始页面上的“Open Project”（打开工程）图标。
2. 遍历至 `project_name.hw` 目录，此目录位于 Vivado IDE 工程目录中。
3. 选择位于 `project_name.hw` 目录中的 `.lpr` 工程文件，然后单击“OK”。
4. 连接至您的硬件。
5. 使用正确的器件镜像文件以及来自相应 Vivado 运行目录的 `.ltx` 文件来进行烧录和调试。
6. 打开工程时，用户首选项、运行时管理器调试仪表板和窗口设置都将复原。

烧录功能

打开工程并将硬件管理器与目标器件相连后，即可在 Vivado Lab Edition 中使用 Vivado Design Suite 所提供的所有烧录功能。所有烧录相关 Tcl 命令在 Vivado Lab Edition 中都受支持。如需获取有关可用烧录功能的更多详细信息，请参阅“配置存储器器件烧录”。

相关信息

配置存储器器件烧录

调试功能

打开工程并将硬件管理器与目标器件相连后，即可在 Vivado Lab Edition 中使用 Vivado Design Suite 所提供的所有调试功能。所有调试相关 Tcl 命令在 Vivado Lab Edition 中都受支持。如需了解有关可用的调试功能的更多详细信息，请参阅本用户指南的“在硬件中调试逻辑设计”部分。

相关信息

在硬件中调试逻辑设计

第 3 章

生成比特流或器件镜像

在生成比特流或器件镜像之前，请复查其设置，确保这些设置对于您的设计都正确无误，这一点至关重要。

AMD Vivado™ IDE 中的比特流和器件镜像设置分为 2 种类型：

1. 比特流或器件镜像文件格式设置。
2. 器件配置设置。

在 Vivado Flow Navigator 中依次选择“Settings ” → “Bitstream” （设置 > 比特流），或者选择“Flow” → “Settings” → “Bitstream Settings” （流程 > 设置 > 比特流设置）菜单选项以打开“Bitstream Settings” （比特流设置）弹出窗口（如下图所示）。只要设置正确，即可使用 `write_bistream` Tcl 命令或者使用 Vivado Flow Navigator 中的“Generate Bitstream” （生成比特流）按钮来生成比特流数据文件。

如果以 AMD Versal™ 器件为目标，则会生成可编程器件镜像 (`.pdi`)，而不是比特流文件。更改器件镜像设置的过程与先前架构类似，但菜单选项、Tcl 命令和可用设置会有所不同。要访问器件镜像设置，请依次选中 Vivado Flow Navigator 中的“Settings” → “Generate Device Image” （设置 > 生成器件镜像），或者选中“Flow” → “Settings” → “Generate Device Image Settings...” （流程 > 设置 > 生成器件镜像设置）菜单选项，这样即可在“Settings” （设置）弹出窗口中打开“Device Image” （器件镜像）部分（请参阅下图）。要生成器件镜像数据文件，可使用 `write_device_image` Tcl 命令，或者使用 Vivado Flow Navigator 中的“Write Device Image” （写入器件镜像）按钮。如需了解有关 PDI 格式的更多详情，请参阅《Bootgen 用户指南》(UG1283)。

图 5：**比特流设置面板**

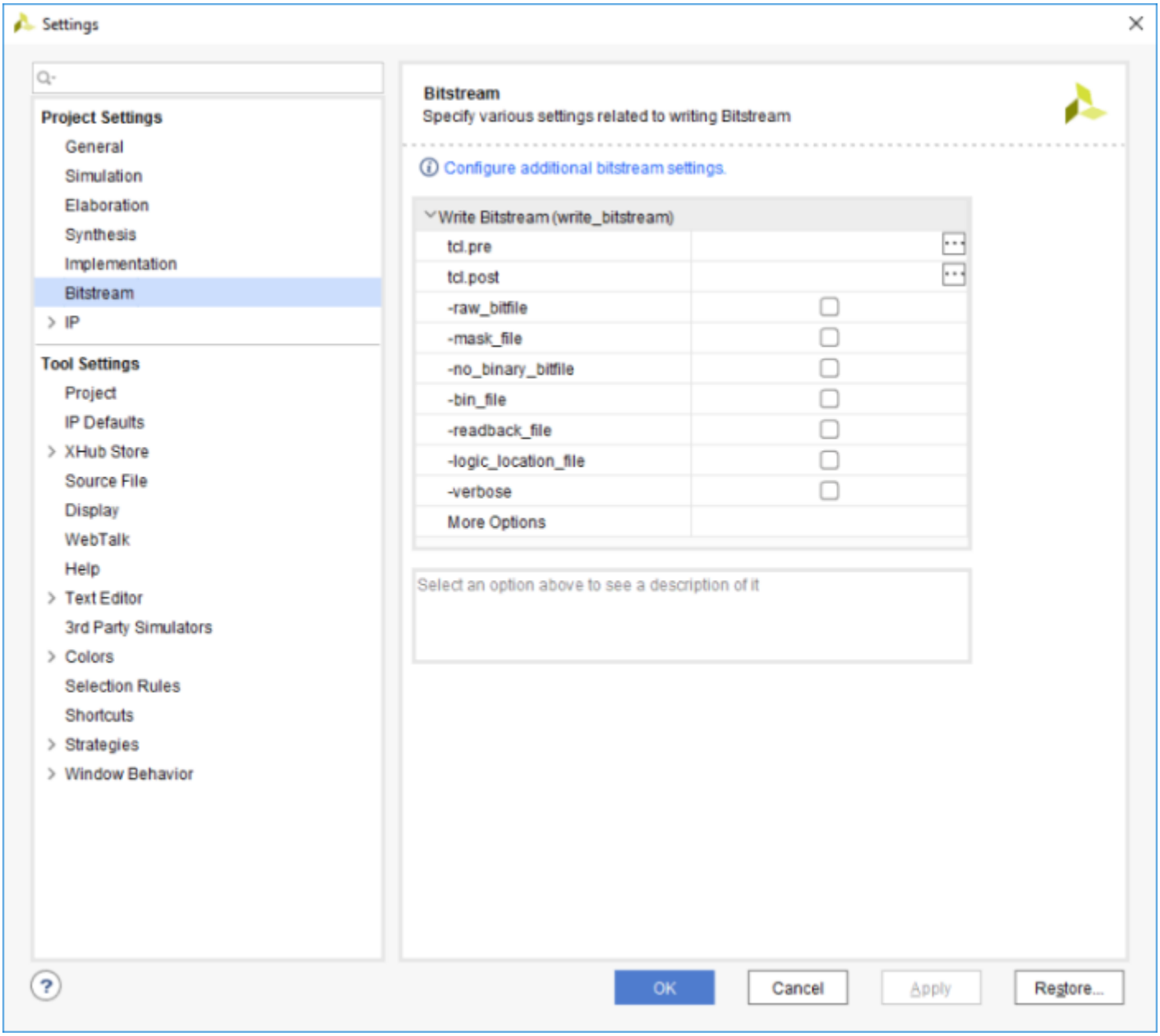
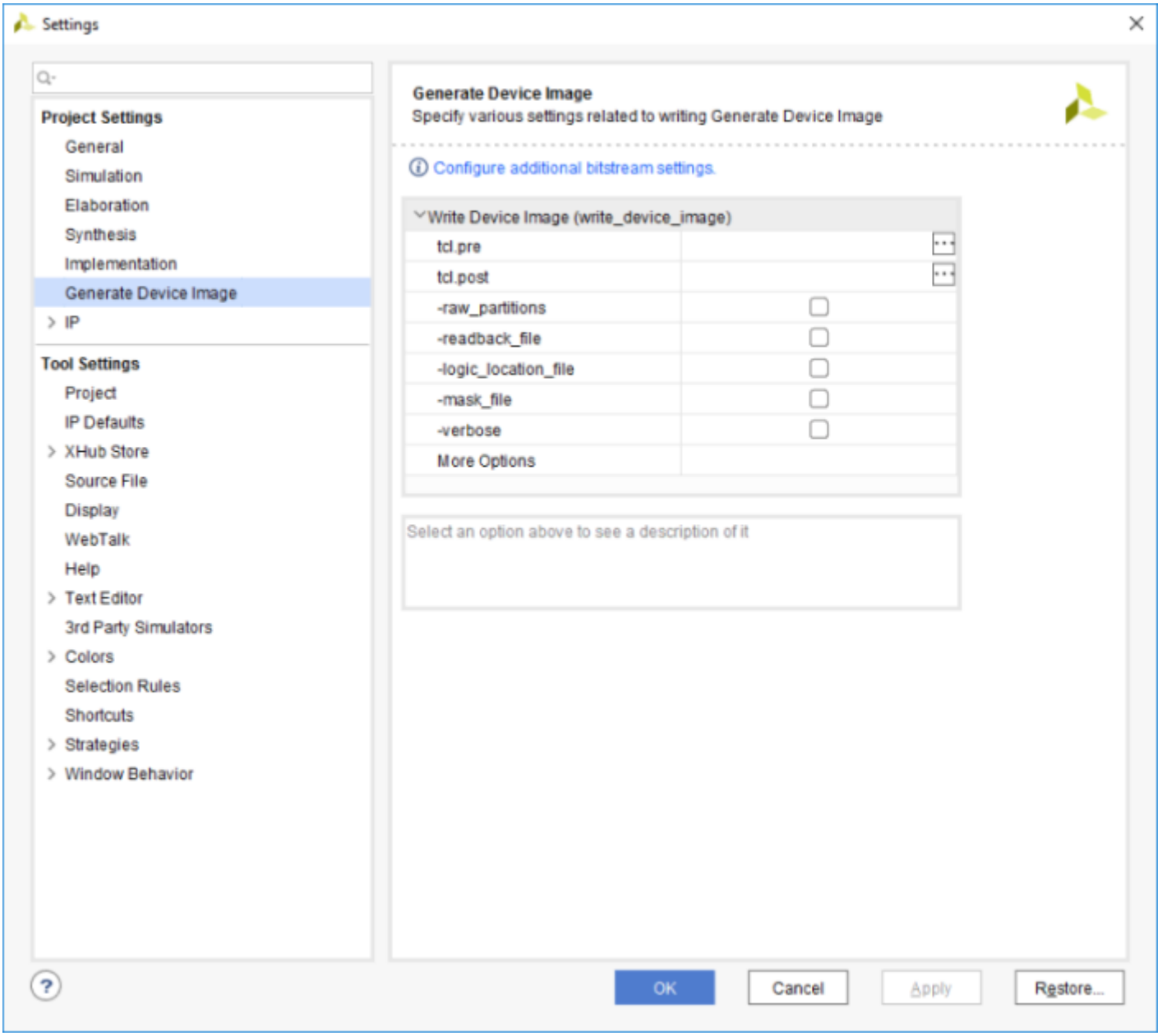

Settings
Project Settings
General
Simulation
Elaboration
Synthesis
Implementation
Bitstream
IP
Tool Settings
Project
IP Defaults
XHub Store
Source File
Display
WebTalk
Help
Text Editor
3rd Party Simulators
Colors
Selection Rules
Shortcuts
Strategies
Window Behavior
Bitstream
Specify various settings related to writing Bitstream
Configure additional bitstream settings.
Write Bitstream (write_bitstream)
tcl.pre
tcl.post
-raw_bitfile
-mask_file
-no_binary_bitfile
-bin_file
-readback_file
-logic_location_file
-verbose
More Options
Select an option above to see a description of it
OK
Cancel
Apply
Restore...

图 6：**生成器件镜像设置面板**



更改比特流文件格式设置

默认情况下，`write_bitstream` Tcl 命令仅生成二进制比特流 (`.bit`) 文件。（可选）您可通过使用以下命令开关来更改 `write_bitstream` Tcl 命令写出的文件格式：

- `-raw_bitfile`：（可选）此开关会导致 write_bitstream 编写原始比特文件 (`.rbt`)，其中所含信息与二进制比特流文件中所含信息相同，但格式为 ASCII。输出文件名为 `<filename>.rbt`。

- `-mask_file`：（可选）编写掩码文件 (`.msk`)，其中包含有关比特流文件中配置数据所在位置的掩码数据。此文件可用于判定比特流中哪些位应与回读数据进行比较和验证。如果掩码位为 0，那么应根据比特流数据验证该位。如果掩码位为 1，那么不应验证该位。输出文件名为 `<file>.msk`。
- `-no_binary_bitfile`：（可选）不编写二进制比特流文件 (`.bit`)。如果要生成 ASCII 比特流文件或掩码文件或者要生成比特流报告（而不生成二进制比特流文件），请使用此命令。
- `-logic_location_file`：（可选）创建 ASCII 逻辑位置文件 (`.ll`)，以显示锁存器、触发器、LUT、块 RAM 和 I/O 块输入输出的比特流位置。这些位元可供位置文件中的帧和位编号引用，以帮助您观察 FPGA 寄存器的内容。
- `-bin_file`：（可选）创建二进制文件 (`.bin`)，其中仅包含器件烧录数据，不含标准比特流文件 (`.bit`) 中找到的报头信息。
- `-reference_bitfile <arg>`：（可选）读取引用比特流文件，并输出增量比特流文件，其中仅含不同于指定引用文件的内容。此部分比特流文件可用于对含更新设计的现有器件进行增量烧录。

更改器件镜像 (PDI) 文件格式设置

默认情况下，`write_device_image` Tcl 命令仅生成 1 个 `.pdi` 文件。（可选）您可通过使用以下命令开关来更改 `write_device_image` Tcl 命令写出的文件格式：

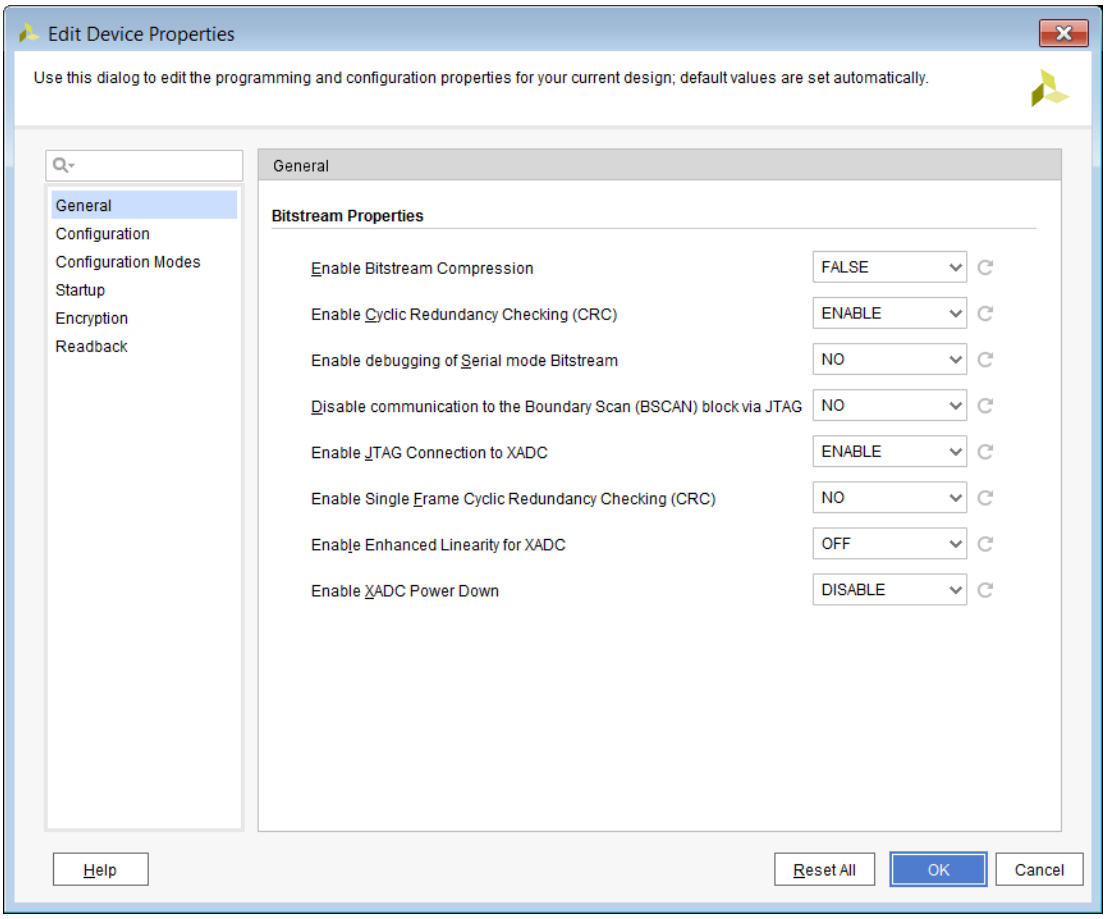
- `-force`（可选）：覆盖现有文件。
- `-verbose`（可选）：打印 write_device_image 选项。
- `-raw_partitions`（可选）：写入原始 CFI 和 NPI 分区文件（`.rnpi` 和 `.rcdo`）
- `-mask_file`（可选）：写入掩码文件 (`.msk`)
- `-logic_location_file`（可选）：写入逻辑位置文件 (`.ll`)
- `-cell <arg>`（可选）：仅为指定单元创建部分器件镜像。
- `-no_pdi`：不生成 pdi 文件。仅生成原始分区文件后即停止操作。
- `-no_partial_pdifile`（可选）：不为 Dynamic Function eXchange 设计写入部分 `pdi` 文件。
- `-quiet`（可选）：忽略命令错误。
- `<file>`（必需）：要写入的 `.pdi` 文件名。

更改器件配置比特流设置

您可更改的最常见的配置设置归为器件配置设置类别。这些设置是器件模型的属性，您可使用“Edit Device Properties”（编辑器件属性）对话框来为选定的已综合或已实现的设计网表更改这些设置。以下步骤描述了如何使用此方法来设置各种比特流属性：

1. 选择“Tools”→“Edit Device Properties”（工具 > 编辑器件属性）。
2. 在“Edit Device Properties”对话框中，选择左侧列中的类别之一（请参阅下图）。

提示：您可在“Search”（搜索）字段中输入属性。例如，在“Search”文本框中输入 jtag 即可查找并选中与 JTAG 烧录相关的属性。



3. 将属性设为期望的值，然后单击“OK”（确定）。
4. 依次选择“File” → “Constraints” → “Save”（文件 > 约束 > 保存）以将更新后的属性保存到目标 XDC 文件中。

您也可以在 XDC 文件中使用 `set_property` 命令来设置比特流属性。例如，以下提供了如何更改 start-up DONE cycle 属性的示例：

```
set_property BITSTREAM.STARTUP.DONE_CYCLE 4 [current_design]
```

在 Vivado 模板中提供了更多示例和模板。“器件配置比特流设置”描述了所有器件配置设置。

重要提示！只需编辑与所用配置模式相关的“器件配置比特流设置”即可。其他设置请保留默认值。

相关信息

器件配置比特流或 PDI 设置

第 4 章

器件烧录

生成器件镜像后，下一步是将其下载到目标器件。Vivado IDE 具有内置原生的系统内器件烧录功能用于执行此操作。

Vivado Design Suite 和 Vivado Lab Edition 都包含相应的功能，支持您连接到包含 1 个或多个 AMD 器件的硬件，以便对这些器件进行烧录并与之交互。要连接到硬件，可通过 Vivado Lab Edition 或 Vivado Design Suite 图形用户界面，也可使用 Tcl 命令。无论采用何种方法，连接到硬件并对目标器件执行烧录的步骤都相同：

1. 打开“Hardware Manager”（硬件管理器）。
2. 打开主机上运行的硬件服务器所管理的硬件目标。
3. 将器件镜像与相应的器件相关联。
4. 将器件镜像烧录或下载到硬件器件中。

打开硬件管理器

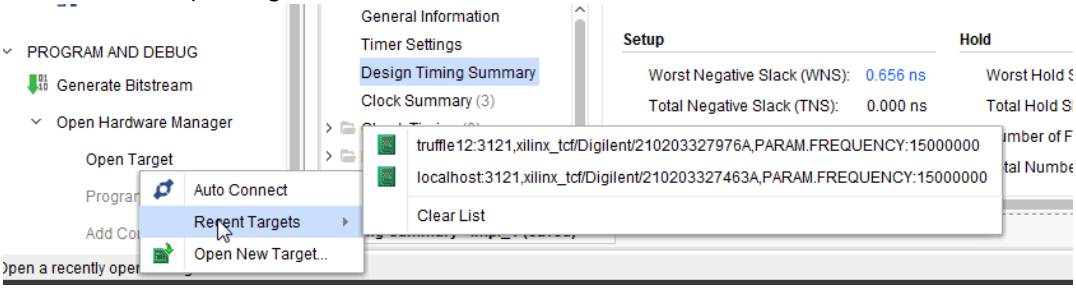
打开硬件管理器是对硬件中的设计进行烧录和/或调试的第一步。要打开硬件管理器，请执行以下操作之一：

- 如果工程已打开，请单击 Vivado Flow Navigator 的“Program and Debug”（烧录和调试）部分中的“Open Hardware Manager”（打开硬件管理器）按钮。
- 选择“Flow”→“Open Hardware Manager”（流程 > 打开硬件管理器）。
- 在 Tcl 控制台窗口中，运行 `open_hw_manager` 命令。

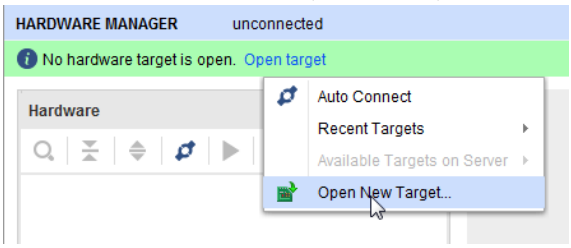
打开硬件目标连接

打开硬件目标（如包含 JTAG 链的硬件开发板，该链中包含一个或多个 FPGA 或自适应 SoC）后，下一步是连接到硬件服务器，它负责管理与硬件目标间的连接。有 3 种方法可用：

- 在 Vivado Flow Navigator 的“Program and Debug”（烧录和调试）部分中的“Hardware Manager”（硬件管理器）下，使用“Open Target”（打开目标）选项打开新的硬件目标或最近的硬件目标（请参阅下图）。



- 使用“Hardware Manager”窗口顶部的绿色用户辅助功能栏上的“Open Target” → “Recent targets”（打开目标 > 最近的目标）或“Open Target” → “Open New Target”（打开目标 > 打开新目标）选项即可分别打开最近的硬件目标或新的硬件目标（如下图所示）。



- 使用 Tcl 命令打开到硬件目标的连接。

提示： “Auto Connect”（自动连接）选项可用于自动连接至本地硬件目标。

使用 hw_server 连接至硬件目标

当 hw_server 连接至本地机器上的目标时，Vivado 会自动将其启动。但您也可以在本地机器或远程机器上手动启动 hw_server。例如，如果在 Windows 平台上完整安装 Vivado，请在 `cmd` 命令提示符处运行以下命令：

```
C:\Xilinx\Vivado\<Vivado_version>\bin\hw_server.bat
```

如果当前在 Windows 平台上使用的是 Hardware Server (Standalone) 安装，那么请在 `cmd` 命令提示符处运行以下命令：

```
c:\Xilinx\HWSRVR\<Vivado_version>\bin\hw_server.bat
```

请遵循下一章节中的步骤，使用此代理打开到新硬件目标的连接。

要获取兼容 JTAG 下载线缆和器件列表，请参阅 附录 D: hw_server 支持的 JTAG 线缆和器件。

如需了解有关使用 SmartLynq 数据线缆的更多信息，请参阅《SmartLynq 数据线缆用户指南》(UG1258)。

重要提示！ 如果 Vivado 硬件管理器已连接至 hw_server 并且 hw_server 已停止，那么硬件管理器会自动检测到此状况，并断开与服务器的连接。

打开新硬件目标

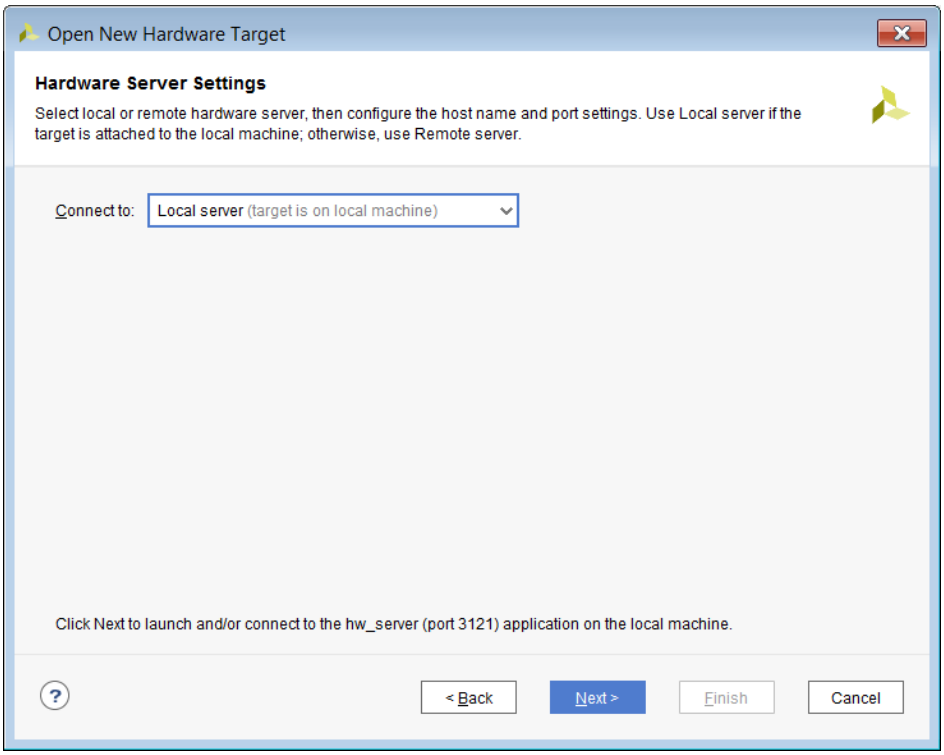
“Open New Hardware Target”Wizard（打开新硬件目标向导）提供了一种交互式方法，供您用于连接到硬件服务器和目标。此向导进程包含以下步骤：

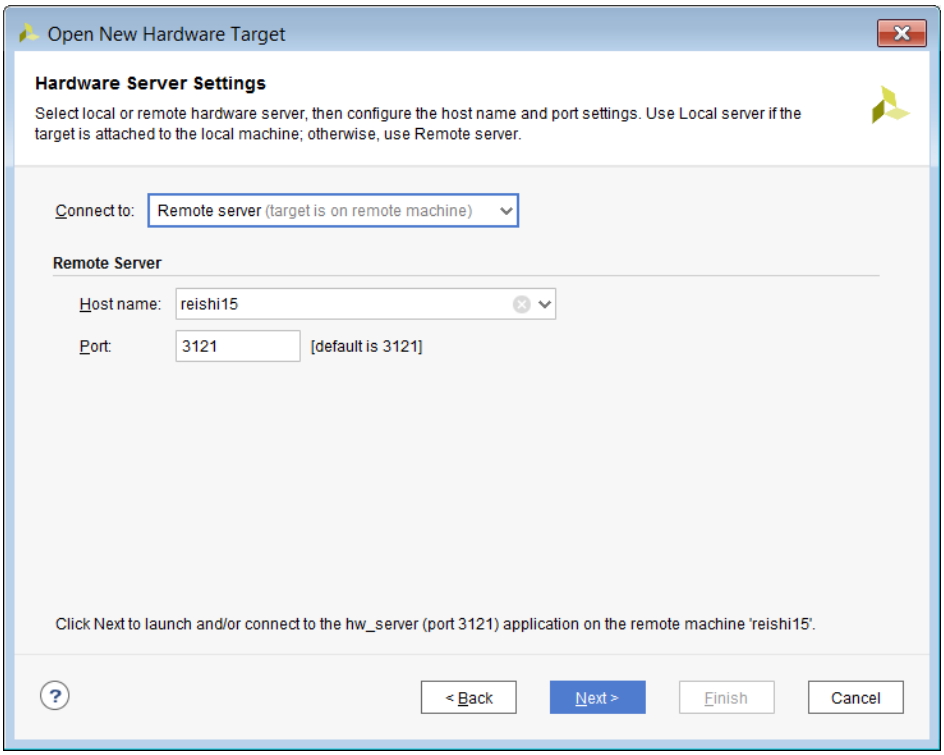
1. 根据您的硬件目标连接到的机器，选中本地服务器或远程服务器：
   - “Local server”（本地服务器）：如果您的硬件目标连接到当前正在运行 Vivado Lab Edition 或 Vivado IDE 的机器，请使用此设置（请参阅下图）。Vivado 软件会在本地机器上自动启动 Vivado Hardware Server (hw_server) 应用。

- “Remote server”（远程服务器）：如果您的硬件目标连接到的机器并非当前正与运行 Vivado Lab Edition 或 Vivado IDE 的机器，请使用此设置。指定远程机器的主机名或 IP 地址以及该机器上运行的 Hardware Server (hw_server) 应用的端口号（请参阅下图）。请参阅“连接到实验室机器上运行的远程 hw_server”以获取有关远程调试的更多信息。

重要提示！使用远程服务器时，您需要手动启动 Vivado Hardware Server (`hw_server`) 应用，此应用的版本与您用于连接到硬件服务器的 Vivado 软件版本相同。

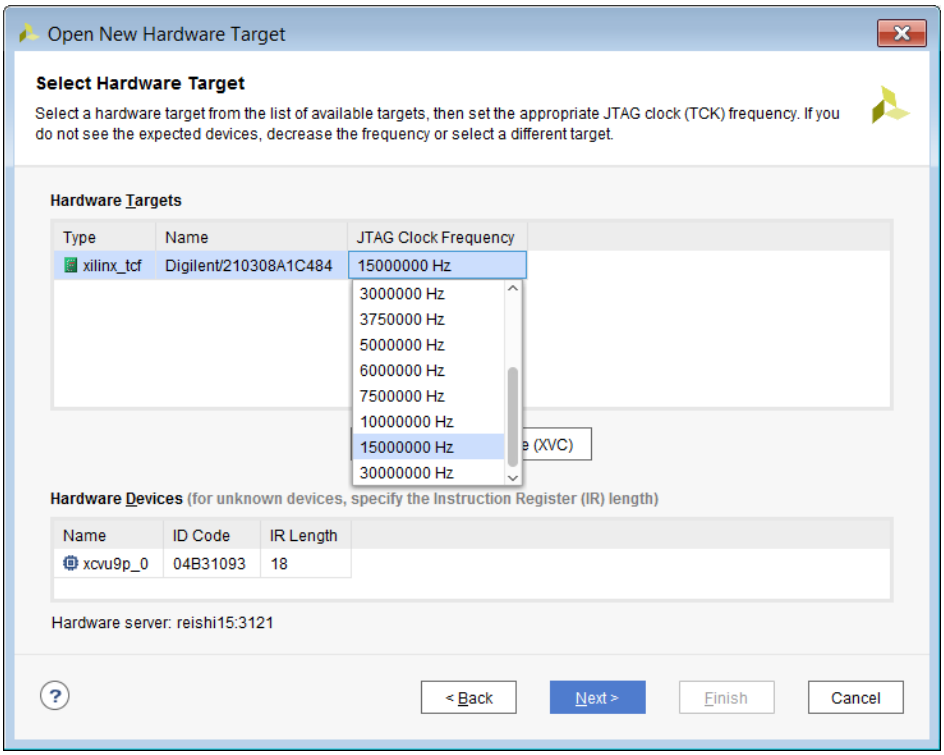
提示：如果只需远程连接到实验室机器，则无需在该远程机器上安装完整的 Vivado Design Suite。可改为在该远程机器上安装轻量级 Vivado Hardware Server (Standalone) 工具。





2. 从硬件服务器管理的目标列表中选择相应的硬件目标。

   注释：选择目标时，您会看到该硬件目标上可用的各种硬件器件。



重要提示！如果在 JTAG 链中存在第三方器件，请使用答复记录 61312 中的指示信息来为未知器件添加 IDCODE、IR 长度和名称。

相关信息

连接到实验室机器上运行的远程 hw_server

对硬件目标进行故障排除

尝试连接至硬件目标时可能会遇到问题。以下是常见问题以及有关解决此类问题的建议：

- 如果您无法正确识别目标上的硬件器件，这可能表示您的硬件无法按默认目标频率运行。您可调整硬件目标或电缆的 TCK 管脚频率（请参阅上图）。

  注释：每一种类型的硬件目标都可能包含不同属性。请参阅每个硬件目标的文档，以了解有关这些属性的更多信息。

- 虽然 Vivado 硬件服务器会尝试自动判定 JTAG 链中所有器件的指令寄存器 (IR) 长度，但在一些极为罕见的情况下，它可能无法正确判定该长度。您应检查每个未知器件的 IR 长度，确保其正确无误。如需指定 IR 长度，可在“Open New Hardware Target”Wizard（打开新硬件目标向导）的“Hardware Devices”（硬件器件）表中直接指定其长度（请参阅“打开新硬件目标”）。

相关信息

打开新硬件目标

打开最近的硬件目标

“Open New Hardware Target”Wizard（打开新硬件目标向导）同样可用于填充先前已连接的硬件目标列表。除使用此向导连接到硬件目标外，您也可以通过选中“Hardware Manager”（硬件管理器）窗口中的“Open recent target link”（打开最近的目标链路），然后选中列表中任一最近连接的硬件服务器/目标组合来重新打开到先前已连接的硬件目标的连接。您还可通过 Vivado Flow Navigator 的“Program and Debug”（烧录和调试）部分中的“Hardware Manager”（硬件管理器）下的“Open Target”（打开目标）选项来访问此最近使用的目标列表。

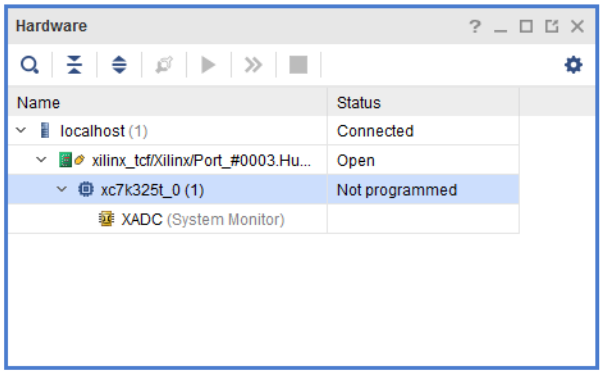
使用 Tcl 命令打开硬件目标

您也可以使用 Tcl 命令来连接至硬件服务器/目标组合。例如，要连接至由 localhost 3121 上运行的 hw_server 所管理的 digilent_plugin 目标（序列号 210203339395A），请使用以下 Tcl 命令：

```
connect_hw_server -url localhost:3121 current_hw_target [get_hw_targets */
xilinx_tcf/Digilent/210203339395A] set_property PARAM.FREQUENCY 15000000
[get_hw_targets \ */xilinx_tcf/Digilent/210203339395A] open_hw_target
```

完成打开到硬件目标的连接后，“Hardware”（硬件）窗口中将显示打开的目标的硬件服务器、硬件目标和各种硬件器件（请参阅下图）。

图 7：**打开到硬件目标的连接后显示的硬件视图**



将烧录文件与硬件器件相关联

连接到硬件目标后，在进行器件烧录之前，需将比特流数据烧录文件与该器件相关联。在“Hardware”（硬件）窗口中选择硬件器件，确保“Properties”（属性）窗口中的“Programming file”（烧录文件）属性设置为相应的烧录文件。

注释：为方便起见，对于已打开的硬件目标中首个匹配器件的“Programming File”属性，Vivado IDE 会自动使用当前实现的设计的烧录文件来作为其属性值。仅当使用 Vivado IDE 工程模式时，此功能才可用。使用 Vivado IDE 非工程模式时，您需要手动设置该属性。

您也可以使用 `set_property` Tcl 命令来设置硬件器件的 `PROGRAM.FILE` 属性：

```
set_property PROGRAM.FILE {C:/<path_to_programming_file>} [lindex
[get_hw_devices] 0]
```

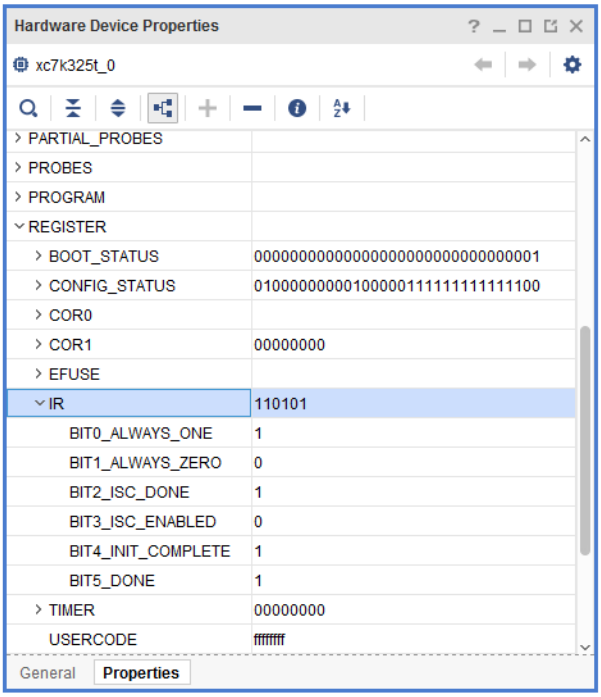
硬件器件烧录

当烧录文件与硬件器件相关联后，即可在“Hardware”（硬件）窗口中右键单击器件并单击“Program Device”（器件烧录）菜单选项来执行硬件器件烧录。您也可以使用 `program_hw_device` Tcl 命令。例如，要对 JTAG 链中的首个器件执行烧录，请使用以下 Tcl 命令：

```
program_hw_devices [lindex [get_hw_devices] 0]
```

当进度对话框指示烧录进度达 100% 完成时，您可通过在“Hardware Device Properties”（硬件器件属性）窗口中检验状态是否为“DONE”（完成）来检查硬件器件烧录是否成功。

图 8：**检查器件的 DONE 状态**



您还可使用 `get_property` Tcl 命令来检查 DONE 状态。例如，要检查 JTAG 链中的首个器件（即 AMD Kintex™ 7 器件）的 DONE 状态，请使用以下 Tcl 命令：

```
get_property REGISTER.IR.BIT5_DONE [lindex [get_hw_devices] 0]
```

在 Versal 器件上使用的命令与此处命令略有不同，因为 DONE 状态寄存器不同。索引值为 1 必须解读为 `get_hw_devices` 返回的首个器件，并且在单器件用例中，将表示为 `arm_dap`。

```
get_property REGISTER.JTAG_STATUS.BIT[34]_DONE [lindex [get_hw_devices] 1]
```

如果您使用其他方法执行硬件器件烧录（例如，使用闪存器件或外部器件烧录器），那么也可以通过右键单击“Refresh Device”（刷新器件）菜单项或者通过运行 `refresh_hw_device` Tcl 命令来刷新硬件器件的状态。这样可以刷新器件的各种属性，包括但不限于 DONE 状态。

重要提示！对于非 Versal 架构，如果您的设计包含调试核，请确保 Debug Hub 时钟频率至少是 JTAG 时钟频率的 2.5 倍。

重要提示！“User SCAN Chain”（用户扫描链）：对于非 Versal 架构，默认情况下，Vivado 烧录器会尝试检测设计中指定的用户扫描链上的调试核。它通过向器件发出 JTAG_CHAIN 1 命令来执行检测。如果您已烧录的器件所含的设计没有任何调试核或者所含调试核的用户扫描链为 2、3 或 4，那么您会看到 1 条警告。

要判定用户扫描链设置，对于非 Versal 架构，请打开已实现的设计并使用：

```
get_property C_USER_SCAN_CHAIN [get_debug_cores dbg_hub]
```

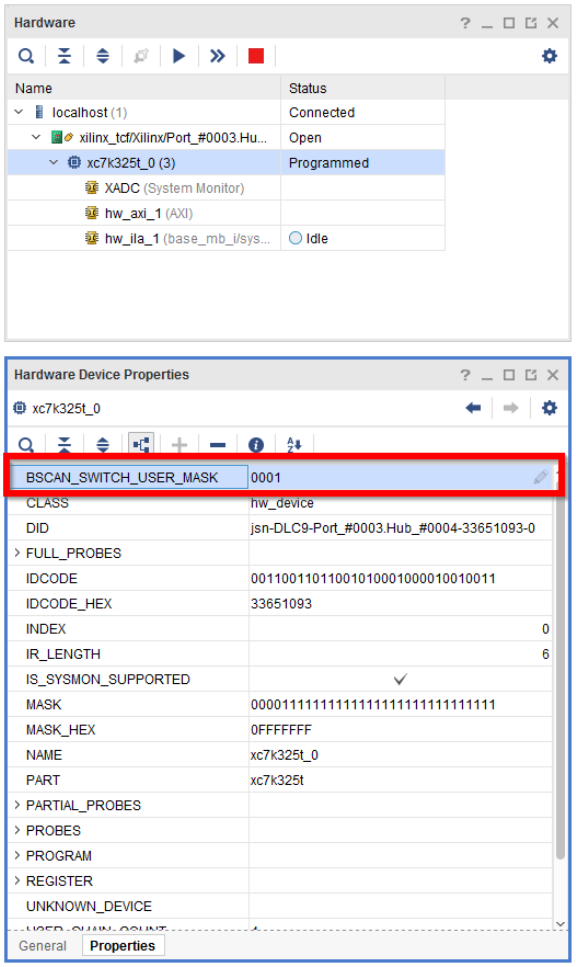
您可在 Vivado 硬件管理器中更改用户扫描链。

注释： `BSCAN_SWITCH_USER_MASK` 是比特掩码值。请参阅下图。

或者，您可指定用户扫描链值作为 hw_server 启动选项。

```
hw_server -e "set bscan-switch-user-mask <user-bit-mask>"
```

图 9：**BSCAN 开关用户掩码**



提示：对于早于 Vivado 2016.3 的设计，AMD 建议手动启动 hw_server 并包含 -e"set xsdb-user-bscan <C_USER_SCAN_CHAIN scan_chain_number>"，以检测位于用户扫描链的 2 或 4 上的 Debug Hub。

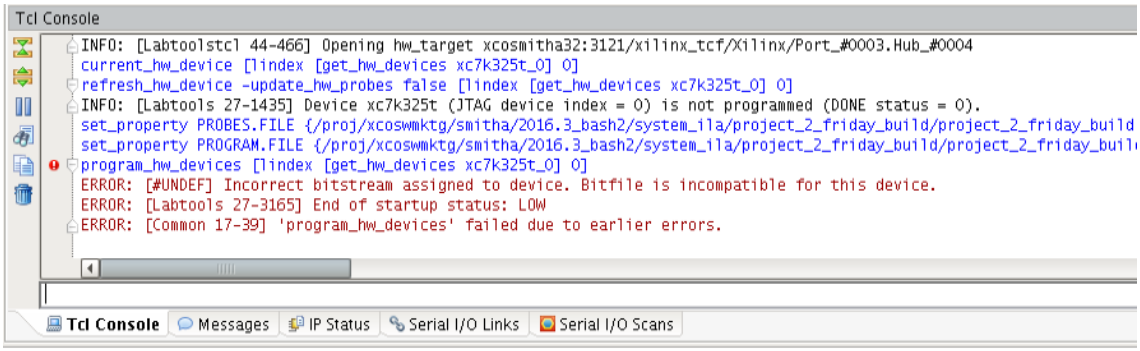
比特流分配错误消息

Vivado 硬件管理器在以下情况下会显示“incorrect bitstream assignment”（比特流分配错误）消息：

- 烧录镜像时，尝试使用为其他 FPGA 或自适应 SoC 生成的比特流或可编程器件镜像来进行烧录。

  例如，尝试使用 XCVU190 比特流烧录 XCKU115 时，会显示以下错误消息。

图 10：**使用 XCVU190 比特流烧录 XCKU115**



解决方案是为要烧录的 FPGA 或自适应 SoC 指定正确的比特流。

尝试对连接到 FPGA 器件的配置存储器进行烧录

为了对连接到 FPGA 器件的配置存储器进行烧录，Vivado 硬件管理器首先将闪存控制器比特流下载至 FPGA 器件。硬件管理器通过 FPGA 器件的 JTAG 端口发送闪存命令和数据以供控制器进行处理，此控制器会将经过处理的闪存命令/数据发送至配置存储器接口。

硬件管理器所下载的控制器比特流是专为 FPGA 器件的最新硅片版本而生成的。例如，2016.3 中的 XCKU115 的配置存储器控制器比特流是为后续 XCKU115-es2 硅片生成的。

对连接到此 FPGA 的配置存储器进行烧录时，如果用户开发板上具有 XCKU115-es1 器件，那么将显示“尝试使用针对当前 FPGA 器件的其他硅片版本生成的比特流来烧录当前器件”中所示的错误消息。这是因为硬件管理器正在尝试将 -es2 闪存控制器比特流下载到 -es1 器件中。

关闭硬件目标

您可在“Hardware”（硬件）窗口中右键单击硬件目标并从弹出菜单中单击“Close Target”（关闭目标）来关闭硬件目标。您还可使用 Tcl 命令关闭硬件目标。例如，要关闭本地主机服务器上的 xilinx_platformusb/USB21 目标，请使用以下 Tcl 命令：

```
close_hw_target {localhost/xilinx_tcf/Digilent/210203339395A}
```

重要提示！如果开发板已掉电或者电缆已断开连接，那么 Vivado IDE 会关闭硬件管理器中的硬件目标。同时还会取消 Vivado 主线程中的任意 Vivado 操作。如果开发板已重新上电或者电缆已重新连接，那么 Vivado IDE 会尝试重新打开硬件管理器中的硬件目标。

关闭到硬件服务器的连接

您可在“Hardware”（硬件）窗口中右键单击硬件服务器并从弹出菜单中选择“Close Server”（关闭服务器）来关闭硬件服务器。您还可使用 Tcl 命令关闭硬件服务器。例如，要关闭到 localhost 服务器的连接，请使用以下 Tcl 命令：

```
disconnect_hw_server localhost
```

重要提示！如果 Vivado 硬件管理器已连接至 hw_server 并且 hw_server 已停止，那么硬件管理器会自动检测到此状况，并断开与服务器的连接。

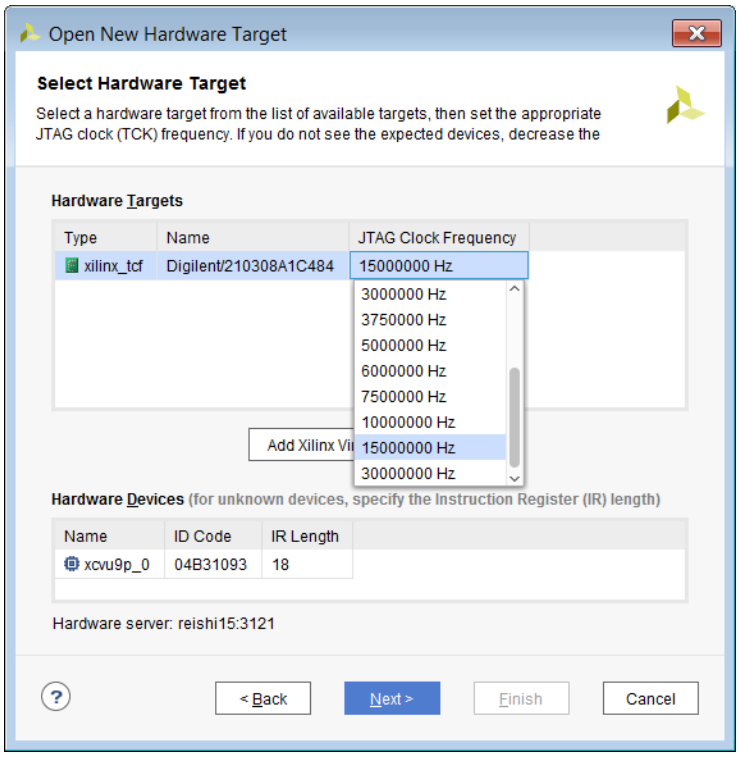
以更低的 JTAG 时钟频率重新连接到目标器件

JTAG 链的速度受链中最慢的器件所限。因此，要降低 JTAG 时钟频率，请连接到所设 JTAG 时钟频率低于默认 JTAG 时钟频率的目标器件。

您应尝试以默认 JTAG 时钟频率打开连接，针对 Digilent 线缆连接，默认频率为 15 MHz，针对 USB 线连接则为 6 MHz。如果无法以上述速度建立连接，AMD 建议您进一步降低默认 JTAG 时钟频率，如下所示。

要更改 JTAG 时钟频率，请使用 AMD Vivado™ Design Suite 中的“Open New Hardware Target”Wizard（打开新硬件目标向导），如下图所示。

图 11：**Vivado 降低 JTAG 频率**



或者，您可以按顺序使用下列 Tcl 命令：

```
open_hw_manager
```

```
connect_hw_server -url machinename:3121
```

```
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/210203327962A]
```

```
set_property PARAM.FREQUENCY 250000 [get_hw_targets */xilinx_tcf/Digilent/
210203327962A] open_hw_target
```

连接到 JTAG 链中包含超过 32 个器件的服务器

在 Vivado 中，可连接的服务器的 JTAG 链中可包含超过 32 个器件。您需要在启动 hw_server 时提供 `max-jtag-devices` 选项，才能启用在单一扫描链中检测更多器件的功能。此设置的默认值为 32。

注释：增大该数值会导致器件发现进程变慢，从而导致电缆接入速度变慢。

在 `hw_server` 启动时指定 `max-jtag-devices` 选项的方式如下：

```
hw_server -e "set max-jtag-devices 64"
```

用法

该选项用于启动 hw_server，并且能够启用大于 64 位的 ir 长度。此设置的默认值为 64。对于 JTAG 链中 ir 长度范围更大（例如，93）的器件，可以增大该值。

注释：增大该数值会导致器件发现进程变慢，从而导致电缆接入速度变慢。因此，仅限 ir 长度较长和器件数量较多的系统才能增大该值。

在 `hw_server` 启动时指定该选项的方式如下：

```
hw_server -e "set max-ir-length 93"
```

初始化 (Init) 选项

您还可使用 `--init=script.txt` 选项，通过文件来加载此设置。要使用 `init` 选项，请创建初始化脚本，如以下示例所示。在此脚本中，指定 `set max-jtag-device` 参数。

```
# Sample script.txt set max-ir-length 93
```

启动 hw_server，如以下示例所示：

```
hw_server --init=script.txt
```

更改默认 SmartLynq 端口

默认情况下，SmartLynq 模块使用下列端口。在某些情况下，您可能要更改所使用的端口。要更新所使用的端口，请使用《SmartLynq 数据线缆用户指南》(UG1258) 中记述的过程将这些命令添加至 SmartLynq `config.ini` 文件中并执行更新。

表 1：**更改 SmartLynq 默认端口**

默认端口	描述	添加至 `config.ini` 以执行更改
80	基于 HTTP 的 TCF	`set http-port <port>`
3121	TCF	`set tcf-port <port>`
10200	基于 XVC 的低级别 JTAG 访问	`set xvc-port <port>`
3000-3005	GNU 调试器端口（适用于 Arm®/ MicroBlaze™ 处理器）	`set gdb-port <base port>` 设置基本端口即可为最多 4 个端口设置最低的端口。

第 5 章

利用 pdi_dbg_util 调试 PDI 烧录

烧录 AMD Versal™ 器件时，您可能遇到烧录错误。器件的启动 ROM 和 Platform Loader and Manager (PLM) 软件都可能会报告这些错误。要理解和分析这些启动错误可能并不容易，因为这些错误都经过编码，可能需要读取其他寄存器才能识别错误的根本原因。

pdi_dbg_util 实用工具可用于辅助启动配置错误的解码和分析。该工具会执行错误值查找、PDI log 日志分析和 PDI 烧录与分析。通过使用 pdi_dbg_util，您即可高效收集有关烧录启动配置错误的所有必要信息，以便解决 Versal 问题。

以下是组成 pdi_dbg_util 的子命令：

- decode 子命令
- analyze-log 子命令
- analyze-hw 子命令
- program 子命令
- analyze-hw 示例

本章旨在解释这些命令的用途及其使用方法。

PDI 错误解码

pdi_dbg_util 最基本的错误是对 PDI 错误进行解码。发生启动配置错误时，您可以使用 JTAG 或者从 Versal 器件的 UART 输出来获取 PDI 错误信息。

decode 子命令

decode 命令用于对一个或多个 ROM 错误或 PLM 错误进行解码。pdi_dbg_util decode 子命令的实参如下所示：

```
pdi_dbg_util decode [OPTIONS] ERROR_NUMBERS...
```

其中选项包括：

- -c, --category [rom|plm]
  - 错误类别 [默认值：plm]
- -a, --architecture [versal|versal_net|versalnet]
  - 器件架构 [默认值：Versal]

错误编号通常由主错误和次错误组成。您可提供一个或多个错误编号以供解码。如果工具发现一个匹配的错误编号，它会显示该错误的名称和描述。

decode 示例

以下显示了 Versal 器件的 PDI log 日志输出摘录示例。在此示例中，Versal 器件的 PDI log 日志输出生成了 PLM 错误状态 0x28010007：

```
[1057.942]CMD Payload START, Len:0x000000030x00000000F2000038: 0x2940C000
0x00000004 0x00000001
[1066.558]CMD Payload END[1068.886]PLM Error Status: 0x28010007
```

要明确此错误的含义，请调用 pdi_dbg_util，如下所示：

```
pdi_dbg_util decode -a versal -c plm 0x28010007
```

这会生成如下输出：

```
PLM Major Error: 0x2801
------------------------------------------------
Name: CDO_CMD_ERR_EM_SET_ACTION
Description: Set Error Action

PLM Minor Error: 0x0007
------------------------------------------------
Name: XPLMI_INVALID_NODE_ID
```

PLM 错误状态 0x28010007 由 PLM 主错误 0x2801 和 PLM 次错误 0x0007 组成。这两个错误的组合有助于缩小启动期间检测到的具体错误的范围。

使用 PDI 日志分析错误

发生启动配置错误时，您可从 Versal 器件的 UART 或 JTAG 获取 PDI 启动 log 日志消息。通过 Vivado 的硬件管理器，您可运行以下命令以获取当前器件的 log 日志：

```
set logFile [open "pdi.log" w]; puts $logFile [get_property PROGRAM.LOG
[current_hw_device]]; close $logFile
```

这将生成一个名为 `pdi.log` 的文件，供 `pdi_dbg_util analyze-log` 命令使用。

通过 xsdb，您可运行以下命令以获取类似的日志：

```
set logFile [open "xsdb_pdi.log" w]; puts $logFile [device status -hex
jtag_status]; puts $logFile [device status -hex error_status]; plm log -
handle $logFile; close $logFile
```

在此情况下，它会生成 `xsdb_pdi.log` 文件，此文件可搭配 `pdi_dbg_util analyze-log` 命令一起使用。

analyze-log 子命令

analyze-log 子命令用于分析 PDI 配置错误日志。pdi_dbg_util `analyze-log` 命令的实参如下所示：

```
pdi_dbg_util analyze-log [OPTIONS] LOG
```

其中选项包括：

- -p, --pdi_file FILE
  - PDI 文件路径
- -a, --architecture [versal | versal_net]
  - 器件架构 [默认值：Versal]
- -d, --device TEXT
  - 器件名称 [默认值：xcvc1902]

LOG 实参是 PLM log 日志文件，此文件是由 Vivado 硬件管理器或 xsdb 生成的，或者是从器件的 UART 终端捕获的。

analyze-log 示例

以下提供了运行 pdi_dbg_util `analyze-log` 子命令的示例。在此示例中，PDI 的目标器件是属于 Versal 架构的 xvc1902。

```
pdi_dbg_util analyze-log -p design_top.pdi -a versal -d xcvc1902 pdi.log
```

此示例的输出如下：

```
Analyzing PLM Log....
[308.467]PLM Error Status: 0x03260014
    PLM Major Error: 0x0326
    ------------------------------------------------
    Name: XLOADER_ERR_GEN_IDCODE
    Description: Error if Vivado configured part (IDCODE) mismatches with
the actual part

    PLM Minor Error: 0x0014
    ------------------------------------------------
    Name: XLOADER_ERR_IDCODE
    Description: IDCODE mismatch
```

在此情况下，日志输出的错误表明存在 IDCODE 不匹配。对于此示例，log 日志是在 xcvm1802 上尝试使用 `design_top.pdi` 启动时生成的。由于此 PDI 与 xcvm1802 器件不兼容，因此启动 log 日志显示配置已终止，并生成错误 0x03260014。

此示例通过 PDI log 日志中显示，analyze-log 命令可以读取并处理该文件中包含的错误。在处理无法直接访问 Versal 器件进行调试的远程用例时，此命令很有用。

就地错误分析

如果可使用 JTAG 访问 Versal 器件，那么 pdi_dbg_util 的 `analyze-hw` 子命令可帮助诊断配置错误。此命令支持连接到处于活动状态的系统、检索当前 PDI log 日志，并访问与遇到的错误相关的其他寄存器。如果系统启动尝试失败，或者在 DFX 用例下，如果启动后镜像配置失败，那么该命令尤其有用。

analyze-hw 子命令

analyze-hw 子命令可通过 JTAG 接口来就地分析配置错误。pdi_dbg_util analyze-hw 子命令的实参如下所示：

```
pdi_dbg_util analyze-hw [OPTIONS]
```

其中选项包括：

- -p, --pdi_file PATH
  - PDI 文件路径
- -c, --cs_url TEXT
  - ChipScope 服务器 URL [默认值：TCP:localhost:3042]
- -t, --target_index INTEGER
  - 目标器件的索引。使用 list-targets 命令获取 JTAG 目标的列表。[默认值：0]
- -u, --url TEXT
  - 硬件服务器 URL [默认值：TCP:localhost:3121]

analyze-hw 子命令具有多种必需标志和可选标志。使用 -p 选项或 --pdi_file 选项即可传入 PDI 文件。-p 选项指定用于配置器件的 pdi_file。您也可以使用 hw_server connection 字符串来设置 -u 选项。目标索引是使用 -t 选项指定的。此目标是 JTAG 链上的目标器件索引，用于错误分析。如果此目标位于索引 0 上，则可省略 -t 标志以使用默认值 0。最后，可选 -c 标志可用于指定 cs_server，用于根据错误类型进行额外的分析。如不提供 cs_server，则该工具会跳过进一步的分析操作。

analyze-hw 示例

在此示例中，pdi_dbg_util 用于连接到“myhost”主机，以便就地分析 Versal 器件。运行子命令 analyze-hw 前，必须先确定目标索引。要确定目标索引，请首先运行以下命令：

```
pdi_dbg_util list-targets -u TCP:myhost:3121
```

运行该命令将显示系统上的所有目标。输出如下所示：

```
Targets available on hardware server TCP:myhost:3121 :

 Index    Device                    JTAG Target
-------  --------  ------------------------------------------
   0     xcvm1802  jsn-VMK180 FT4232H-421952113768A-04caa093-0
```

此输出表明 xcvm1802 器件位于目标索引 0 处。如果 JTAG 扫描链较长，则会显示此链上检测到的所有器件。此信息可用于传入 analyze-hw 子命令的目标索引。

以下显示了 analyze-hw 子命令的示例。对于该示例，已使用 xcvc1902 器件适用的 PDI 对设计进行了配置。但在此示例中，预计设计会失败，因为开发板上的部件是 xcvm1802。在此情况下，“-t”标志的值设为 0。值得注意的是，针对这条 JTAG 链可以省略“-t”选项，因为默认目标索引值为 0。尽管如此，此处仍包含该选项，用以展示目标索引选项的用法。

```
pdi_dbg_util analyze-hw -p design_top.pdi -t 0 -u TCP:myhost:3121
```

此命令生成的输出如下：

```
Analyzing Error Status....

No Errors found for rom
PLM Major Error: 0x0326
------------------------------------------------
Name: XLOADER_ERR_GEN_IDCODE
Description: Error if Vivado configured part (IDCODE) mismatches with the
actual part

PLM Minor Error: 0x0014
------------------------------------------------
Name: XLOADER_ERR_IDCODE
Description: IDCODE mismatch
```

如上所示，`analyze-hw` 子命令无需获取有关目标器件的详细信息。这是因为，`analyze-hw` 子命令会基于选定的目标索引自动检测器件详细信息。此外，这条子命令还负责读取 log 日志并提供额外的状态信息，因此您无需直接提供日志。如果需要更多详细信息来分析这类错误，该命令则会显示有关此问题的更多寄存器信息。

就地烧录和分析

在烧录后要及时处理配置错误，请使用 pdi_dbg_util 的 `program-hw` 子命令。该子命令在检测处于活动状态的系统中的配置错误时的操作类似于 `analyze-hw`。值得注意的是，`program-hw` 提供了额外功能，可以先烧录 PDI，随后在检测到配置错误时检索所有错误状态寄存器和 PDI log 日志。这条集成子命令简化了调试和解决系统配置问题的过程。

program 子命令

program 子命令用于通过 JTAG 接口来就地配置和分析错误。pdi_dbg_util program 子命令的实参如下所示：

```
pdi_dbg_util program [OPTIONS] PDI_FILE
```

其中选项包括：

- --skip_reset
  - 烧录 PDI 前，跳过器件复位
- -c, --cs_url URL
  - ChipScope 服务器 URL [默认值：TCP:localhost:3042]
- -t, --target_index INTEGER
  - 目标器件的索引。使用 list-targets 命令获取 JTAG 目标的列表。[默认值：0]
- -u, --url URL
  - 硬件服务器 URL [默认值：TCP:localhost:3121]

`program` 子命令包含多个实参，与 `analyze-hw` 子命令的实参相似。主参数是用于配置目标器件的 PDI 文件。与“analyze-hw”子命令类似，您必须提供目标索引和硬件服务器 URL。如果在硬件服务器所在的主机上执行此命令，使用默认值 localhost:3121 即可，因此可以省略此选项。同样，当目标索引为 0 时，也可以省略目标索引选项。值得注意的是，若目标索引未知，则可使用“list-targets”子命令来判定。最后，可选择指定 ChipScope 服务器 URL 选项，以便进行深入的错误分析。

program 子命令具有一个特殊选项，名为 –skip_reset。在使用 PDI 文件配置器件前，可选择该选项来避免执行复位。需要摆脱该工具单独进行系统复位时，或者在处理 DFX PDI 配置等特殊情况时，该功能特性很有用。

program 示例

该示例与 `analyze-hw` 子命令所用的示例类似。这里在错误的目标器件上完成了 PDI 烧录。此示例所用命令如下：

```
pdi_dbg_util program -t 0 -u TCP:myhost:3121 design_top.pdi
```

同样，因为目标器件为 0，所以不需要 -t 选项。但在此添加该选项的目的是演示目标索引选项的使用方式。

此示例的输出如下：

```
INFO: Programming device with: design_top.pdi

Device program progress —  21% Aborted
An error occurred during programming: PLM Error Major 0x326 Minor 0x14
occurred during programming.

Analyzing Error Status....

No Errors found for rom
PLM Major Error: 0x0326
-------------------------------------------------
Name: XLOADER_ERR_GEN_IDCODE
Description: Error if Vivado configured part (IDCODE) mismatches with the
actual part

PLM Minor Error: 0x0014
-------------------------------------------------
Name: XLOADER_ERR_IDCODE
Description: IDCODE mismatch
```

在该输出中，`program` 子命令首先用 PDI 文件来配置器件。烧录状态表示进度；在此例中，它达到 21% 时因报告错误而停止。其中呈现了错误状态，并发起了分析进程。分析包括错误解密，而后得出分析总结。

摘要

pdi_dbg_util 具有若干子命令以帮助处理 Versal 配置错误。这些命令汇总如下：

- `decode`：对一个或多个 ROM 错误和 PLM 错误进行解码
- `analyze-log`：分析 PDI 配置错误日志
- `analyze-hw`：通过 JTAG 接口就地分析配置错误
- `program`：通过 JTAG 接口就地配置并分析错误
- `list-targets`：用于列出 JTAG 链上检测到的目标。此命令用于辅助判定“analyze-hw”和“program”子命令的目标索引。

除这些子命令外，pdi_dbg_util 还有下列全局选项，这些选项适用于所有子命令：

- -v, --verbose
  - 启用打印详细信息
- -p, --pretty
  - 为 pdi_dbg_util 输出启用额外修饰和格式化
- --help
  - 显示选项和子命令的简短帮助信息

pdi_dbg_util 提供了一系列全面的分析，用于识别配置错误。根据您对 Versal 器件的访问权限，您可以根据具体用例来使用相关子命令。

第 6 章

在 Vivado 中执行远程调试

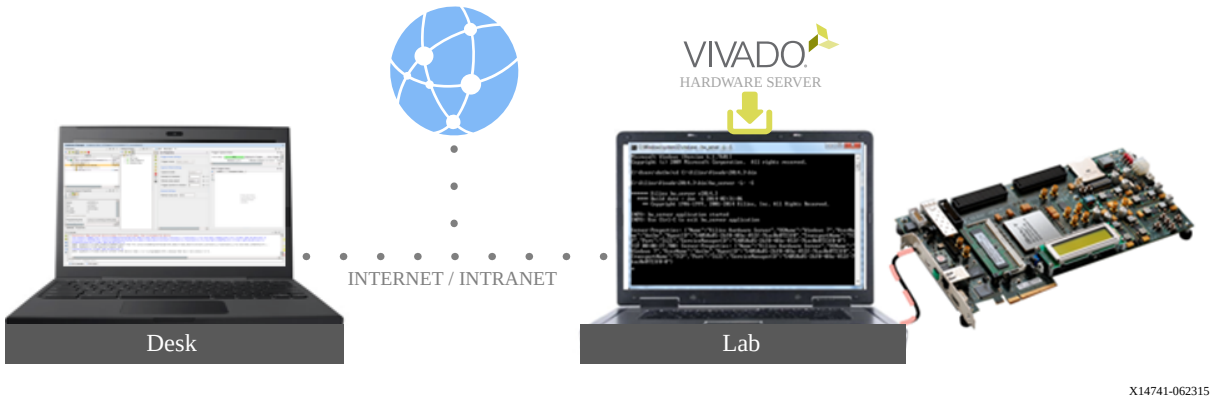
远程调试需求可能在各种情况下出现。在产品原型设计阶段可能需要远程调试以便您在无法实地访问实验室的情况下对实验室中的设计进行调试，或者供您在组织内部共享资源。执行现场调试以诊断问题或者延长产品生命周期时也可能需要远程调试。

AMD 提供多种解决方案来对您的设计进行远程调试。您可使用 AMD 硬件服务器产品连接到实验室中的远程计算机来执行调试。也可以实现 AMD 虚拟线缆 (XVC) 协议来连接到联网开发板。在以下章节中对上述每一种情况进行了详细解释。

使用 Vivado 硬件服务器通过以太网进行调试

您可使用 Vivado 硬件服务器产品来连接到远程实验室机器。该产品为小型 (<100 MB) 独立安装产品，可供安装在实验室机器上。该选项需要内部网或互联网访问权限，也可在您的组织内部使用。

图 12：**使用硬件服务器通过互联网/内部网进行调试**



AMD 虚拟线缆 (XVC)

Vivado IDE 支持 AMD 虚拟线缆 (XVC) 协议。XVC 支持您无需使用 USB 或并行配置线缆即可访问并调试 AMD 器件。此功能有助于 Vivado IDE 对满足下列条件的设计进行调试：

- 设计所含 FPGA 位于无法访问的位置（即，此类位置不方便使用实验室 PC）。
- 无法直接访问开发板上的器件管脚 - 例如，如果 JTAG 管脚只能通过本地微处理器接口来访问。

XVC 是基于互联网的 (TCP/IP) 协议，工作方式与 JTAG 线缆类似。它包含非常基本的电缆命令。这样即可支持 XVC 通过内部网或甚至是通过互联网来调试系统。借助此功能，您可节省差旅成本，或者在差旅不可行情况下仍能解决问题，并缩短远程系统调试所需的时间。

XVC 的另一个常见用途是帮助团队按需访问位于别处的共享系统。如果存在多种物理因素导致系统的使用受到约束（例如，JTAG 连接器不可用或不可访问），也可以使用 XVC。XVC 实现属于编程语言，与平台无关。

您可使用现有以太网连接代替专用 JTAG 报头来创建从处理器到目标器件的相应 JTAG 命令。借助 XVC v1.0 协议，Vivado 即可通过以太网连接与相同的 JTAG 命令进行通信，同时仍可支持所有现有 Vivado 调试功能。

重要提示！如果将 Vivado Debug Bridge IP 用于 XVC，则 Vivado IDE 不支持编程功能。此声明假定器件是在使用 XVC 调试设计之前完成编程的。Debug Bridge IP 与 Versal 自适应 SoC 不兼容。

Vivado Debug Bridge IP 和 AMD 虚拟线缆 (XVC) 流程

注释：在 AMD Versal™ 器件上不支持 Vivado Debug Bridge IP。

Vivado Debug Bridge IP 核属于可提供多个选项的控制器，用于与设计中的调试核进行通信。此设计可采用扁平设计或 Dynamic Function eXchange 设计。此外，Debug Bridge IP 核还可通过配置来利用设计调试，此类设计使用 JTAG 线缆或者通过以太网、PCIe® 或其他接口（无需 JTAG 线缆）来进行远程调试。

Debug Bridge IP 中的多种不同模式可支持多种不同用例。

XVC 模式下的 Debug Bridge

在 Debug Bridge 下有 5 种模式可在 AMD 虚拟线缆 (XVC) 实现中使用。

- From AXI to BSCAN：在此模式下，Debug Bridge 通过 AXI4-Lite 从接口 (slave interface) 来接收 XVC 命令。
- From JTAG to BSCAN：在此模式下，Debug Bridge 通过由用户逻辑驱动的 JTAG 从接口来接收 XVC 命令。
- From PCIe to BSCAN：在此模式下，Debug Bridge 通过 PCIe 扩展配置从接口来接收 XVC 命令。
- From PCIe to JTAG：在此模式下，Debug Bridge 通过 PCIe 扩展配置接口来接收 XVC 命令。此 Debug Bridge 会通过 I/O 管脚使 JTAG 管脚脱离 FPGA。此模式主要用于通过 XVC 对另一块板上的设计进行调试。
- From AXI to JTAG：在此模式下，Debug Bridge 通过 AXI4-Lite 接口来接收 XVC 命令，以便通过 JTAG 管脚将其发送到目标器件。

在上述所有模式下，Debug Bridge 均可通过 Soft-BSCAN（边界扫描）接口与设计中的其他调试核或 Debug Bridge 实例进行进一步通信。Soft BSCAN 主接口支持将 JTAG 接口扩展至内部用户定义的扫描链或 Debug Bridge 实例。

在 Dynamic Function eXchange 设计中使用 Debug Bridge IP

Debug Bridge IP 可在扁平设计内使用，也可在 Dynamic Function eXchange 设计内使用。下面详述了在 Dynamic Function eXchange 设计的静态区域或可重配置分区 (RP) 区域内使用的 Debug Bridge 配置。根据设计要求，在单一分区内允许使用多个 Debug Bridge 实例。

- BSCAN Primitive：当静态区域内需要使用含 BSCAN 原语的 Debug Bridge 时，则使用此模式。此 Debug Bridge 的 BSCAN 主接口可连接到静态区域和/或 RP 区域中的另一个 Debug Bridge 实例，从而提供一条或多条通信路径，以便对这些区域进行调试。

- From BSCAN to Debug Hub：在此模式下，Debug Bridge 使用 BSCAN 从接口来与 Vivado 硬件管理器通信。它使用 Debug Hub 接口来与相关静态区域或 RP 区域中的设计核进行通信。您还可选择向此 Debug Bridge 的输出添加更多 BSCAN 主接口，以便支持对其他调试核（例如，MicroBlaze™ 或 MicroBlaze™ V Debug Module (MDM)）或其他 Debug Bridge 实例进行调试。

  注释：该工具会将 RP 中的调试核自动连接到 Debug Bridge（前提是，它是分区内例化的唯一 Debug Bridge）。

- From AXI to BSCAN：在此模式下，Debug Bridge 通过 AXI4-Lite 从接口来接收 XVC 命令。此 Debug Bridge 还可通过 Soft-BSCAN（边界扫描）主接口与设计中的其他调试核或 Debug Bridge 实例进行进一步通信。Soft BSCAN 接口支持将 JTAG 接口扩展至内部用户定义的扫描链或 Debug Bridge 实例。

- From JTAG to BSCAN：在此模式下，Debug Bridge 通过由用户逻辑驱动的 JTAG 从接口来接收 XVC 命令。此 Debug Bridge 还可通过 Soft-BSCAN（边界扫描）主接口与设计中的其他调试核或 Debug Bridge 实例进行进一步通信。Soft BSCAN 接口支持将 JTAG 接口扩展至内部用户定义的扫描链或 Debug Bridge 实例。

- From PCIe to BSCAN：在此模式下，Debug Bridge 通过 PCIe 扩展配置从接口来接收 XVC 命令。此 Debug Bridge 还可通过 Soft-BSCAN（边界扫描）接口与设计中的其他调试核或 Debug Bridge 实例进行进一步通信。Soft BSCAN 主接口支持将 JTAG 接口扩展至内部用户定义的扫描链或 Debug Bridge 实例。

  注释：此模式仅可用于 UltraScale+ 和 UltraScale 器件架构

- From PCIe to JTAG：在此模式下，Debug Bridge 通过 PCIe 扩展配置接口来接收 XVC 命令。此 Debug Bridge 会通过 I/O 管脚使 JTAG 管脚脱离 FPGA。此模式主要用于通过 XVC 对另一块板上的设计进行调试。

  注释：此模式仅可用于 UltraScale+ 和 UltraScale 器件架构。

- From AXI to JTAG：在此模式下，Debug Bridge 通过 AXI4-Lite 接口来接收 XVC 命令，以便通过 JTAG 管脚将其发送到目标器件。

JTAG 回退支持

基于 XVC 的调试解决方案可配合 AXI 主接口（如 PCIe® XDMA IP）一起使用。如果 AXI 主接口被挂起，或者无法正常运作，则无法在此类情况下进行调试。为了提供基于 JTAG 的回退调试途径（与 XVC 途径并行），AMD 建议以“BSCAN Primitive”（BSCAN 原语）模式来使用 Debug Bridge。“BSCAN Primitive”模式下的 Debug Bridge 可在静态区域内进行例化，其 BSCAN 主接口可连接到另一个 Debug Bridge（已配置为启用 JTAG 回退支持）的 BSCAN 从接口。有 2 种类型的 JTAG 回退支持：

1. 如果要为驻留在 RP 区域内的 Debug Bridge 提供 JTAG 回退，那么您需要启用外部 BSCAN 主接口 JTAG 回退支持。
2. 如果要为驻留在静态区域（或扁平设计）内的 Debug Bridge 提供 JTAG 回退，则应启用内部 BSCAN 主接口 JTAG 回退支持。

MicroBlaze 和 MicroBlaze V Debug Module (MDM) 支持

Debug Bridge 也支持访问 MicroBlaze 和 MicroBlaze V Debug Module (MDM) 以进行调试。MDM BSCAN 从接口输入可连接到支持输出多个 BSCAN 主接口的任意 Debug Bridge 配置模式（例如，从 AXI 到 BSCAN 且其 BSCAN 主接口计数大于 0）。

多个调试树

Debug Bridge IP 支持在单一设计内设置并配置多个独立的调试树。如需在应用内使特定调试逻辑仅对部分用户（例如，系统管理员）可见，而对其他用户隐藏，则可使用多个独立的调试树。此功能支持在独立设计和 Dynamic Function eXchange 设计内设置独立调试树。其中每个独立调试树均可连接到任意受支持的调试核（例如，ILA 和 VIO）

要启用此功能，您需要在相应模式下为要启用的每个调试树例化 1 个 Debug Bridge IP，可用模式为“From AXI to BSCAN”（从 AXI 到 BSCAN）模式或“From PCIe to BSCAN”（从 PCIe 到 BSCAN）模式。例如，在数据中心设计内，有多种类型的用户访问 DUT，您可在客户可见的地址映射内例化 1 个“From AXI to BSCAN”Debug Bridge IP，而在管理员可见的地址映射内例化另一个“From AXI to BSCAN”Debug Bridge IP。

当管理员和/或客户准备好调试设计后，根据其与调试核的通信方式（例如，PCIe 或 JTAG 管脚），管理员和/或客户只需使用 Vivado 硬件管理器按正确的器件偏移连接到 Debug Bridge。如需获取有关在此模式下将 XVC 流程与 PCIe 核以及 Debug Bridge 搭配使用的更多信息，以及获取设计示例，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

下表列出了不同的 Debug Bridge 模式以及这些模式下可用的功能：

表 2：**Debug Bridge 模式**

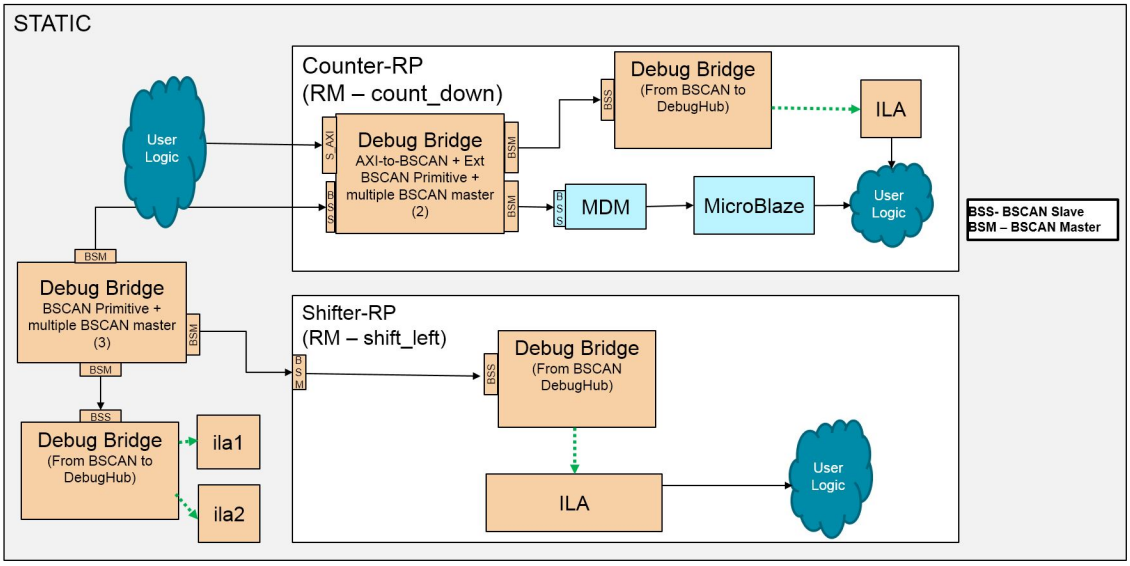
Debug Bridge 模式	XVC 支持	支持在可重配置分区内使用	JTAG 回退支持	MDM 支持
From AXI to BSCAN	支持	支持 [1]	支持 [2]	支持 [3]
From JTAG to BSCAN	支持	支持 [1]	支持 [2]	支持 [3]
From PCIe to BSCAN	支持	支持 [1]	支持 [2]	支持 [3]
From PCIe to JTAG	支持	支持 [1]	不适用	不适用
From BSCAN to DebugHub	不支持	支持 [1]	不适用	支持 [3]
BSCAN Primitive	不支持	不支持	不适用	支持 [3]
From AXI to JTAG	支持	支持	不适用	不适用

注释：

1. “BSCAN Master Count”（BSCAN 主接口计数）可大于 0，并且只能连接到相同 RP 内的其他 Debug Bridge 实例或 MicroBlaze/MicroBlaze V MDM 核。
2. 仅当 Debug Bridge 位于静态分区内时才能使用内部 BSCAN 模式，当 Debug Bridge 位于静态分区或 RP 内时，则可使用外部 BSCAN 模式。
3. “BSCAN Master Count”（BSCAN 主接口计数）可大于 0，并且只能连接到相同 RP 内的其他 Debug Bridge 实例或 MicroBlaze/MicroBlaze V MDM 核。

下图显示了 XVC Debug Bridge 位于 RP 内的设计。

图 13：**Dynamic Function eXchange 设计，其中 XVC Debug Bridge 位于 RP 内**



此 RP 设计含 2 个可重配置分区：计数器 RP 和移位器 RP。此图显示了在静态分区和 RP 区域内使用的不同 Debug Bridge 模式。

此设计的静态分区包含 2 个 Debug Bridge IP。第一个 Debug Bridge IP 处于 BSCAN 原语模式下，并配置为包含 3 个 BSCAN 主接口。其中 2 个 BSCAN 主接口连接到计数器 RP 和移位器 RP 分区内的 Debug Bridge 实例，并提供并行路径用于调试。第三个 BSCAN 主接口则连接到“从 BSCAN 到 Debug Hub”模式下配置的静态分区内的另一个 Debug Bridge 实例。“从 BSCAN 到 Debug Hub”模式下配置的 Debug Bridge 可与设计中的各 Debug IP（ILA、VIO、JTAG-to-AXI 等）进行通信，在此例中它与 ILA IP 进行通信。

在此系统中，计数器 RP 分区包含的 Debug Bridge 是采用“从 AXI 到 BSCAN”模式来例化的。您可在 XVC 模式下使用此 Debug Bridge，此 Debug Bridge 可通过 AXI4-Lite 接口来接收 XVC 命令。此 Debug Bridge 还可通过 Soft-BSCAN（边界扫描）接口与设计中的其他调试 Debug Bridge 实例进行进一步通信。由于此 Debug Bridge 配置为包含 2 个 BSCAN 主接口，因此它与“从 BSCAN 到 Debug Hub”模式下配置的 MDM 和 Debug Bridge 实例进行通信。“从 BSCAN 到 Debug Hub”模式下配置的 Debug Bridge 可与设计中的各 Debug IP（ILA、VIO、JTAG-to-AXI 等）进行通信，在此例中它与 ILA IP 进行通信。

另一方面，移位器 RP 分区仅包含 1 个“从 BSCAN 到 Debug Hub”模式下配置的 Debug Bridge 实例，它可与设计中的各 Debug IP（ILA、VIO、JTAG-to-AXI 等）进行通信，此处它与 ILA IP 进行通信。

如需了解更多信息，请参阅《Debug Bridge LogiCORE IP 产品指南》(PG245)。

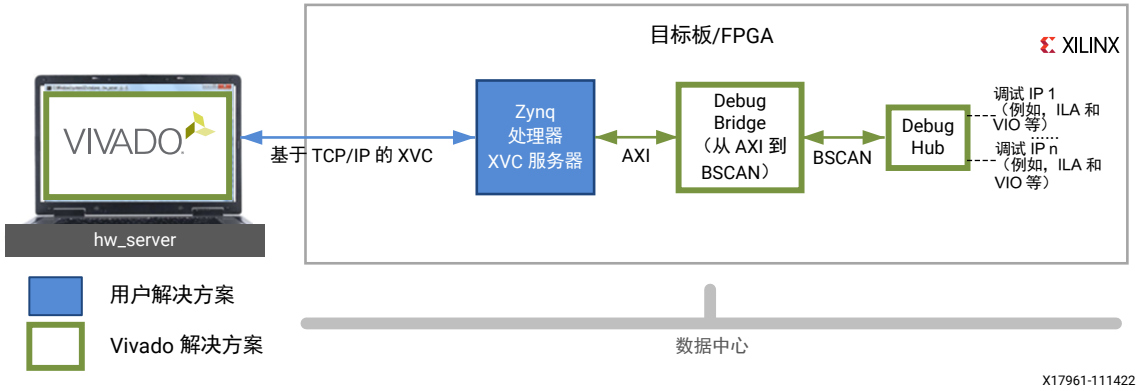
下图显示了部分 Debug Bridge 模式。

From AXI to BSCAN

此桥接类型适用于使用 AMD 虚拟线缆 (XVC) 通过以太网或其他接口远程调试 FPGA 或 SoC 器件（无需 JTAG 线缆）的设计。在此模式下，Debug Bridge 应通过 AXI4-Lite 接口来接收 XVC 命令。此模式用于通过 XVC 对 FPGA 上的设计进行调试。

如需了解更多信息，请参阅《Debug Bridge LogiCORE IP 产品指南》(PG245)。

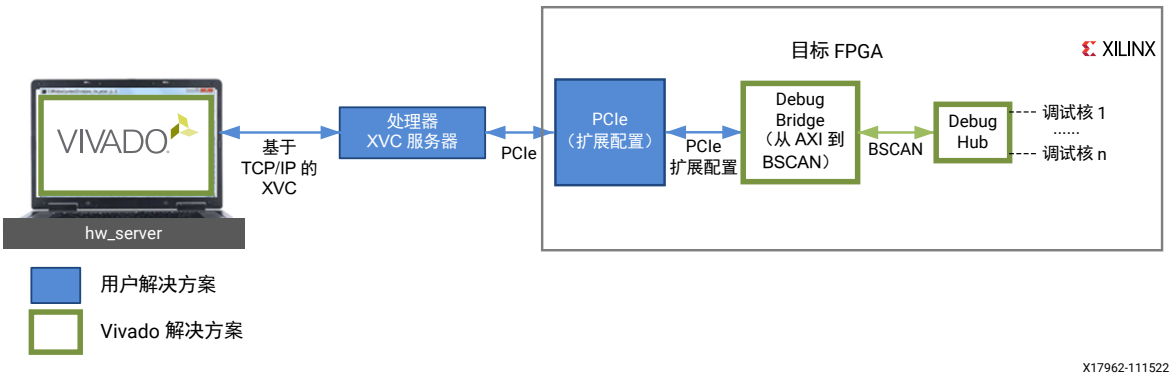
图 14：**AXI 到 BSCAN 模式下的 Debug Bridge**



From PCIe to BSCAN

在典型 PCIe 设置中，您可使用 Debug Bridge 以“PCIe to BSCAN”（从 PCIe 到 BSCAN）模式来与调试核进行通信。在此模式下，Debug Bridge 连接至 PCIe IP 的“Extended Configuration Interface”（扩展配置接口）。这是常见的数据中心用例，其中以 PCIe 作为首选通信路径与主机 PC（而不是 JTAG）进行通信。如需获取有关在此模式下将 XVC 流程与 PCIe 核以及 Debug Bridge 搭配使用的更多信息，以及获取设计示例，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

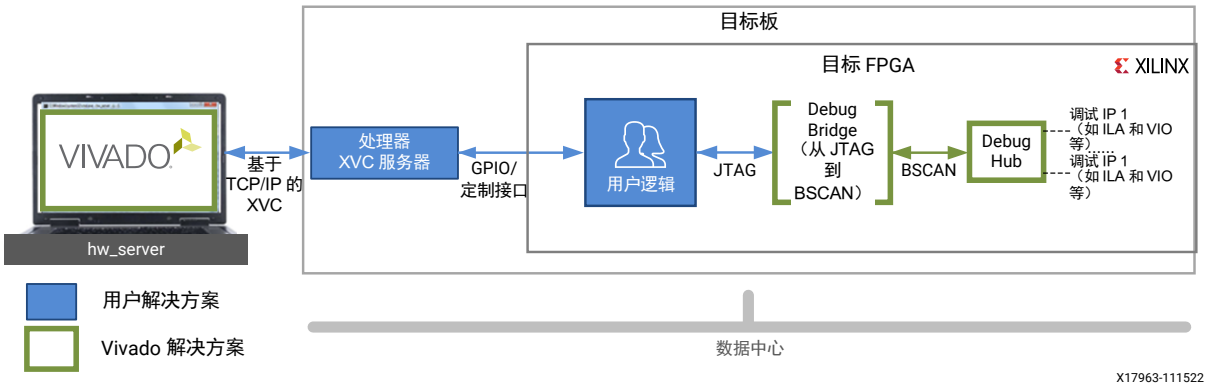
图 15：**PCIe 到 BSCAN 模式下 Debug Bridge 搭配 PCIe 扩展配置接口使用**



From JTAG to BSCAN

此桥接类型适用于使用 AMD 虚拟线缆 (XVC) 通过以太网或其他接口远程调试 FPGA 或 SoC 器件（无需 JTAG 线缆）的设计。在此模式下，Debug Bridge 应通过用户逻辑所驱动的 JTAG 接口来接收 XVC 命令。如需了解更多信息，请参阅《Debug Bridge LogiCORE IP 产品指南》(PG245)。

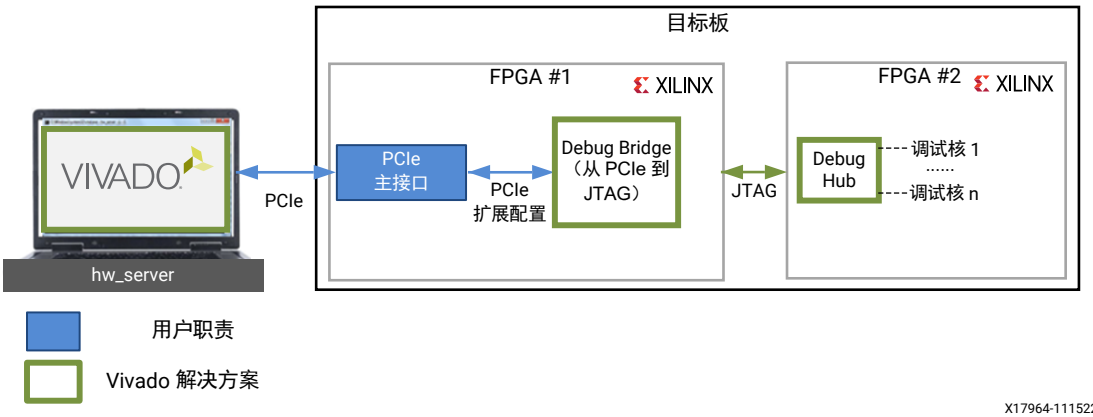
图 16：**JTAG 到 BSCAN 模式下的 Debug Bridge**



From PCIe to JTAG

在 PCIe 设置中，您可使用 Debug Bridge 以“PCIe 到 JTAG”模式来与调试核进行通信。在此模式下，Debug Bridge 连接到 PCIe® IP 的“Extended Configuration Interface”（扩展配置接口），此接口则通过 JTAG 与另一目标 FPGA 上的 Debug Hub 进行通信。

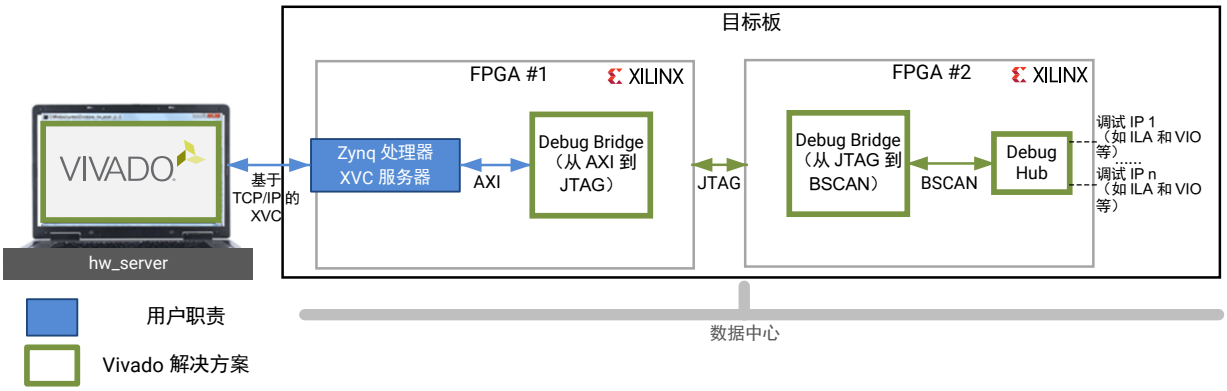
图 17：**PCIe® 到 JTAG 模式下 Debug Bridge 搭配 PCIe 扩展配置接口使用**



From AXI to JTAG

此桥接类型适用于使用 AMD 虚拟线缆 (XVC) 通过以太网或其他接口远程调试 FPGA 或 SoC 器件的设计。在此模式下，Debug Bridge 通过 AXI4-Lite 接口来接收 XVC 命令，以便通过 JTAG 管脚将其发送到目标器件。如需了解更多信息，请参阅《Debug Bridge LogiCORE IP 产品指南》(PG245)。

图 18：**AXI 到 JTAG 模式下的 Debug Bridge**



适用于 Versal 器件的 AMD 虚拟线缆 (XVC) 流程

在 AMD Versal™ 器件上同样支持使用 XVC 作为仅限软件的解决方案，它在 APU 上作为 Linux 应用来运行，无需其他 IP。欲知详情，请访问 AMD 虚拟线缆 GitHub 仓库，链接位于：https://github.com/Xilinx/XilinxVirtualCable。

注释：当前在 Versal 器件上仅支持将 PL 调试核与 XVC 搭配使用。PL 调试核的示例包括 AXIS-ILA 和 AXIS-VIO。Versal 硬核块调试核（如 SYSMON、DDRMC Calibration Debug、PCI® Express Link Debug 和 IBERT）当前不支持通过 XVC 进行远程访问。

XVC 服务器实现

您需要实现 XVC 协议才能在相应的处理器上创建 XVC 服务器。

XVC 协议

XVC 协议允许 Vivado IDE 通过以太网向嵌入式系统发送 JTAG 命令，以便对目标 AMD 器件进行烧录和/或调试。这样即可采用任意供应商解决方案来对 AMD 器件进行调试和烧录。烧录功能所含支持与传统 JTAG 连接所提供的支持相同。调试功能支持操作 Xilinx System Debugger (XSDB) 或 Vivado 硬件调试 IP。

在此情况下发送至器件的 JTAG 命令与使用烧录电缆或使用 Digilent 模块进行本机通信时传输至器件的命令相同。这样可确保该功能在所有现有 Vivado 硬件调试工具之间都可正常运行。

用户 XVC 1.0 命令

下表中汇总了各项 XVC 1.0 协议命令：

表 3：**XVC 命令描述**

命令	描述

getinfo	命令格式：
	`getinfo:`
	此命令用于获取 XVC 服务版本。服务收到“getinfo:”时即运行以下字符串
	`xvcServer_v1.0:<xvc_vector_len>\n`
	`<xvc_vector_len>` 为可移位至服务中的矢量的最大宽度。

表 3：**XVC 命令描述** (续)

命令	描述
shift	命令格式： `shift:[num bits][tms vector][tdi vector]` 此命令可使用字节矢量 `tms_vector` 和 tdi_vector 来移入 `num_bits` `num_bits` 为小字节序模式下的整数。 它表示将矢量移出所需的 TCK clk 翻转数。 `tms_vector` 为字节大小的矢量，含所有 TMS 移位。 该矢量的字节 0 中的位 0 将首先被移出。 该矢量为 `num_bits` 并向上舍入到最近的字节。 `tdi_vector` 与 `tms_vector` 类似，但它表示要移入的所有 tdi 矢量。 此命令返回的字节矢量大小与 `tms_vector` 相同，并针对移入的每个位对相应的 tdo 位进行采样。 该矢量的字节 0 中的位 0 是从移位读取的首个 tdo 值
settck	命令格式： `settck:[period]` 此命令会尝试将服务 tck 周期设置为 `[period]`。 `[period]` 以 ns 为单位来指定。 它是小字节序整数值 此命令完成 `settck:` 后会返回应用的周期。 返回的值以 ns 为单位来指定。 它是小字节序整数值

初始化 Vivado IDE hw_server

通过 XVC 连接来初始化 Vivado IDE hw_server 时，Vivado IDE 会发现 XVC 线缆，就像发现任何 USB 线一样。为此，请使用以下实参来启动 Vivado IDE hw_server：

```
hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200"
```

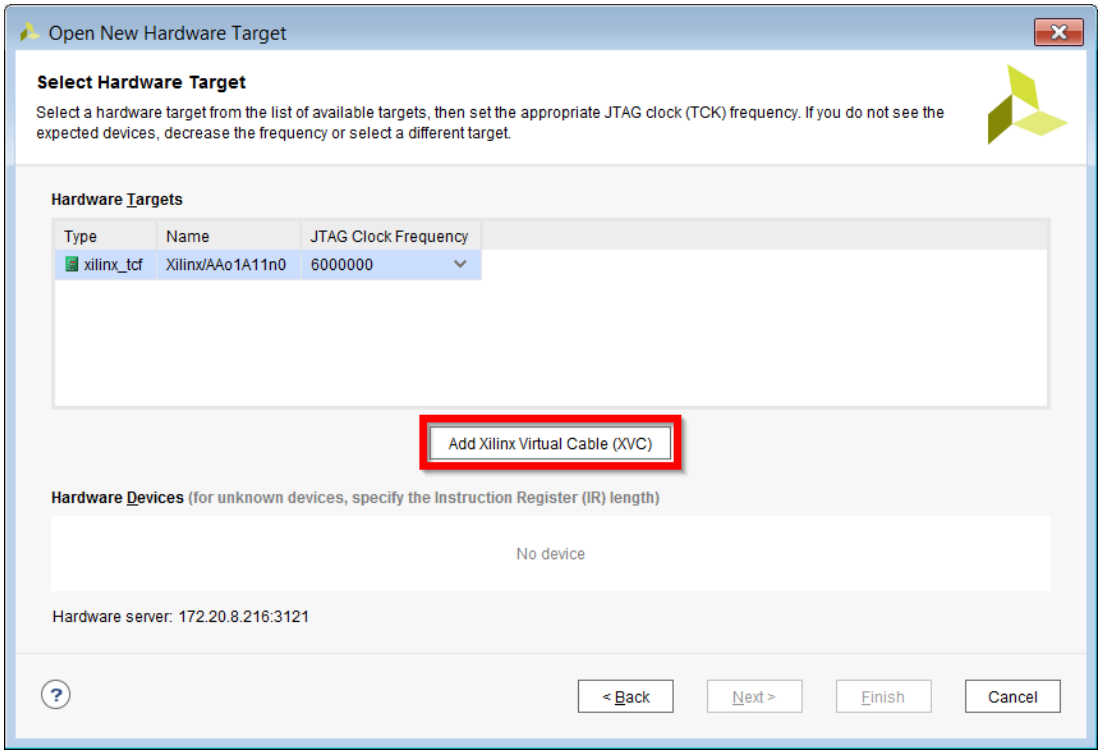
`auto-open-servers` 选项支持由 hw_server 在启动时初始化 XVC 线缆。您可以初始化硬件服务器，以强制连接至现有 XVC 线缆。服务器会在后续连接中自动发现 XVC 线缆。

auto-open-servers 的实参如下所示：

```
xilinx-xvc:<xvc_host_name>:<xvc_port>
```

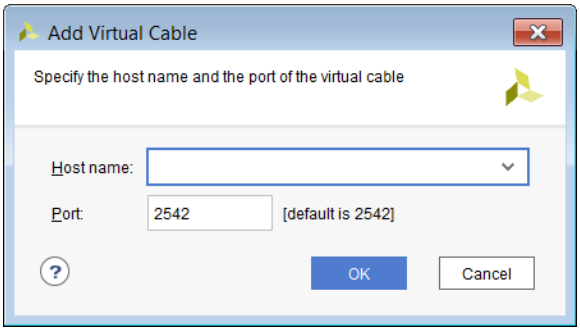
使用逗号分隔字符串即可指定多个服务器。当硬件服务器启动时，它会尝试与指定 XVC 服务器建立连接。或者，您也可以在使用 Vivado 硬件管理器的“Open New Hardware Target”Wizard（打开新硬件目标向导）连接到目标时提供 XVC 服务器详细信息，如下图所示。

图 19： **“Open New Hardware Target”对话框**



单击“Add Xilinx Virtual Cable”（添加 AMD 虚拟线缆）按钮。这样将显示“Add Virtual Cable”（添加虚拟线缆）对话框，如下图所示。

图 20： **“Add Virtual Cable”对话框**



提供要连接到的 XVC“Hostname”（主机名）和“Port number”（端口号）。

请参阅《在 Zynq 7000 上使用 PetaLinux 工具运行 AMD 虚拟线缆》(XAPP1251) 以获取相关示例。

本应用指南显示了如何通过使用 PetaLinux 工具生成的 Linux 操作系统来获取在 AMD Zynq™ 7000 器件上运行的 XVC 服务器。其中提供了适用于 Avnet MicroZed 开发板的参考设计。本应用指南中的目标器件为 AC701 开发板，并且该器件将由在 Linux 上运行 XVC 的 MicroZed 开发板来进行烧录和调试。

注释：要获取 XVC 服务器实现（基于 TCP/IP）示例，请参阅以下 GitHub 仓库：https://github.com/Xilinx/XilinxVirtualCable。

第 7 章

配置存储器器件烧录

AMD Vivado™ 器件烧录器功能支持您直接通过 JTAG 对 AMD 器件进行烧录。Vivado 还可通过 JTAG 对基于闪存的选定配置存储器器件进行间接烧录。具体操作是首先使用特殊配置对 AMD FPGA 进行编程，在 JTAG 与闪存器件接口之间提供数据路径，然后使用此数据路径对配置存储器器件内容进行烧录。

Vivado 器件配置功能支持您使用 AMD 或 Digilent 电缆来直接配置 AMD 器件或存储器器件。请参阅“使用 hw_server 连接至硬件目标”，以获取适用电缆的列表。Vivado 可采用边界扫描模式运行，以便对 AMD 器件和配置存储器器件进行配置或烧录。

请参阅“hw_server 支持的 JTAG 线缆和器件”，以获取受 Vivado 支持的配置存储器器件的完整列表。

要在 Vivado 中对配置存储器器件进行烧录并从中启动，请执行以下步骤：

1. 生成比特流镜像，用于配置存储器器件。
2. 创建配置存储器文件（`.mcs` 或 `.bin`）。
3. 连接到 Vivado 中的硬件目标。
4. 添加配置存储器器件。
5. 使用 Vivado IDE 对配置存储器器件进行烧录。
6. 启动 AMD 器件（可选）。

相关信息

使用 hw_server 连接至硬件目标
配置存储器支持

更改器件镜像属性

在已综合或已实现的设计中，依次选择“Tools” → “Edit Device Properties”（工具 > 编辑器件属性）以打开“Edit Device Properties”对话框。

在已综合或已实现的设计中，从 Flow Navigator 内选中“Settings” → “Bitstream”（设置 > 比特流），在 AMD Versal™ 器件上，则选中“Settings → Device Image”（设置 > 器件镜像），然后单击“Configure additional Bitstream Settings”（配置其他比特流设置）链接以打开“Edit Device Properties”（编辑器件属性）对话框，如下所示。

图 21：**Edit Device Properties：FPGA 器件的比特流属性**

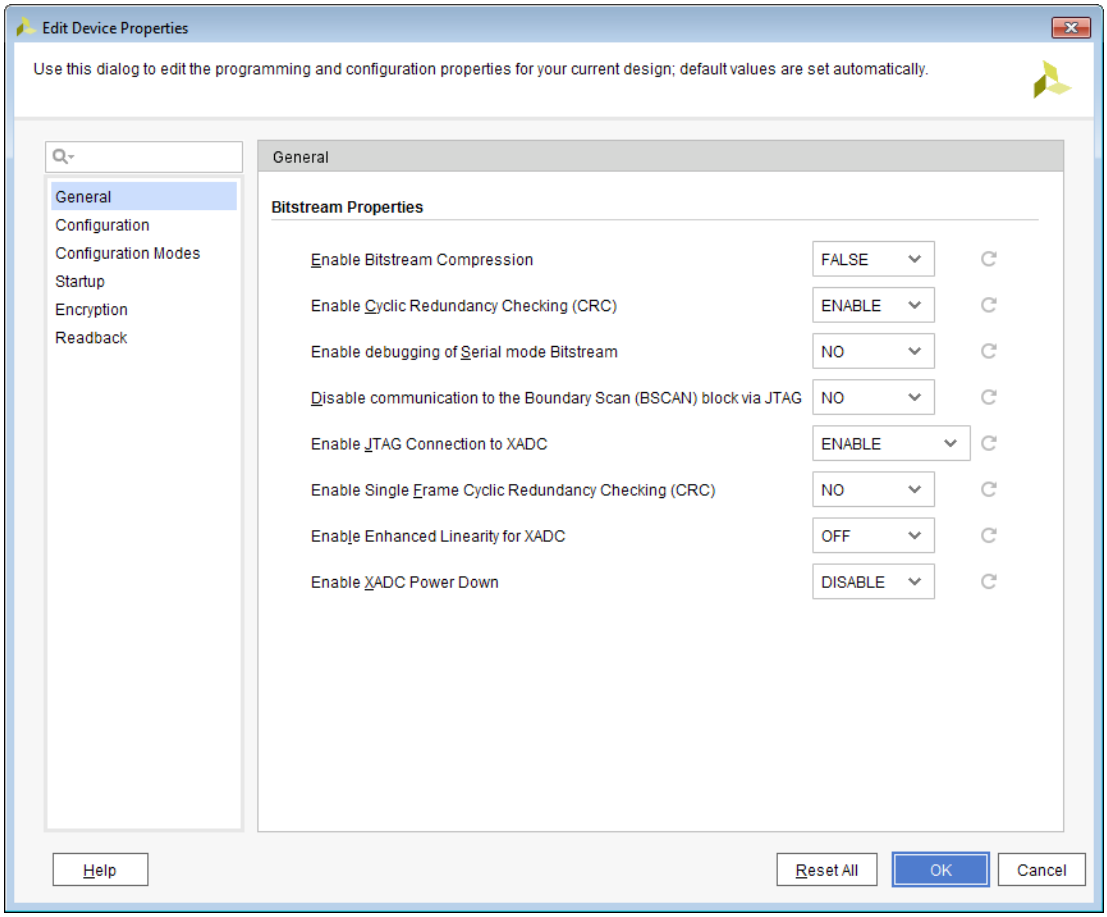
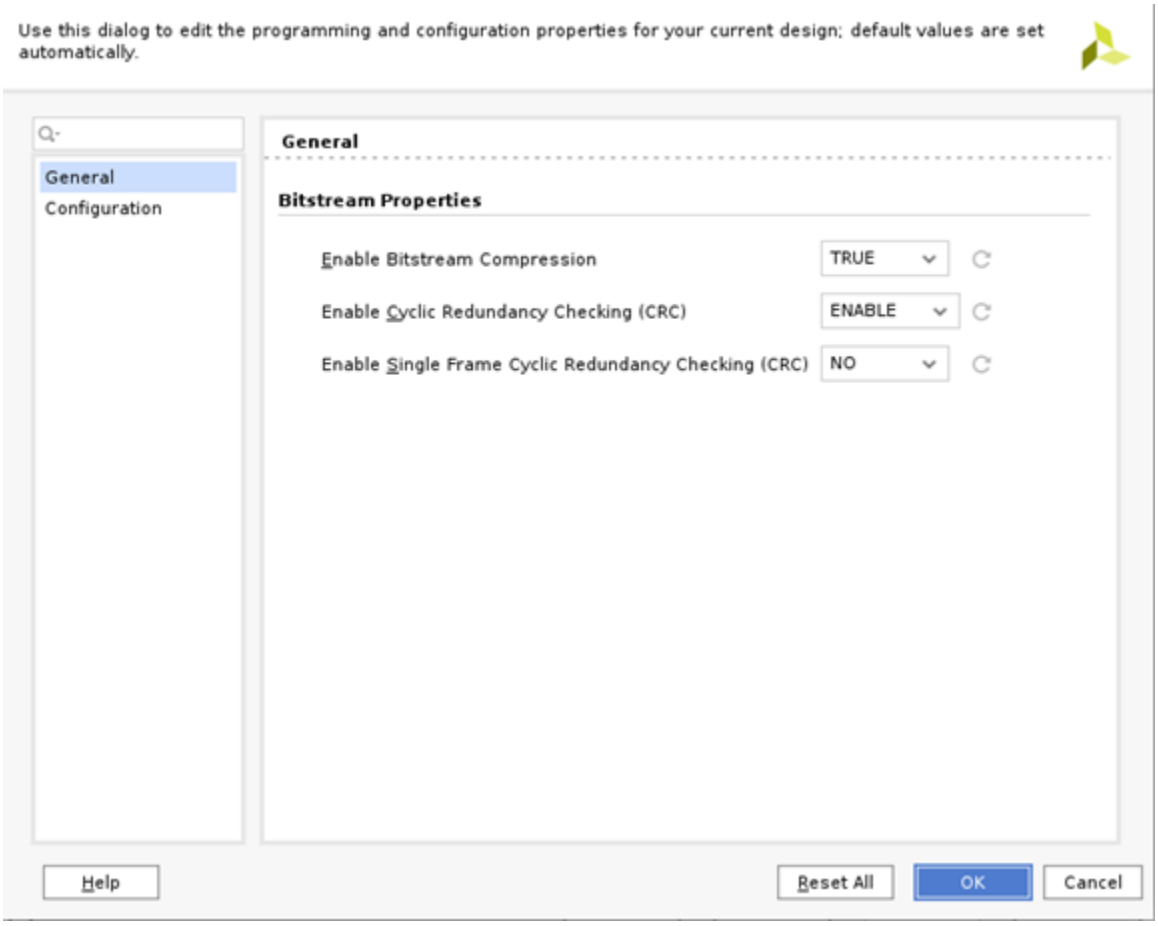


图 22：**Edit Device Properties：Versal 器件的比特流属性**



使用该对话框左上角的搜索字段即可搜索所有 SPI 或 BPI 相关字段并选择相应的选项设置。请参阅“器件配置器件镜像或 PDI 设置”以了解有关器件配置设置的更多信息。

相关信息

器件配置比特流或 PDI 设置

创建配置存储器文件（适用于 FPGA 器件）

使用 write_cfgmem Tcl 命令来创建 `.mcs` 或 `.bin` 烧录文件。此文件用于对配置存储器器件进行烧录。

例如，要生成 `.mcs` 文件以配置具有单个 1 Gbit BPI 配置存储器器件的 FPGA，请执行以下操作：

```
write_cfgmem -format mcs -interface bpix16 -size 128      \
             -loadbit "up 0x0 design.bit"-file design.mcs
```

注释： `write_cfgmem` 的 `-size` 实参以兆字节 (MB) 为单位，不同于基于兆位 (Mb) 的闪存器件容量。因此，大小为 1 Gbit 的闪存器件换算为 128 MB 提供给以上示例中的 `write_cfgmem`。

注释： `write_cfgmem` 会根据比特流大小来自动调整配置存储器文件的大小。

Vivado IDE 支持使用 `write_cfgmem` 命令将多个 `.bit` 文件链接在一起。要为包含多个比特流的单个 1 Gbit BPI 配置存储器器件生成 `.mcs` 文件，请执行以下操作：

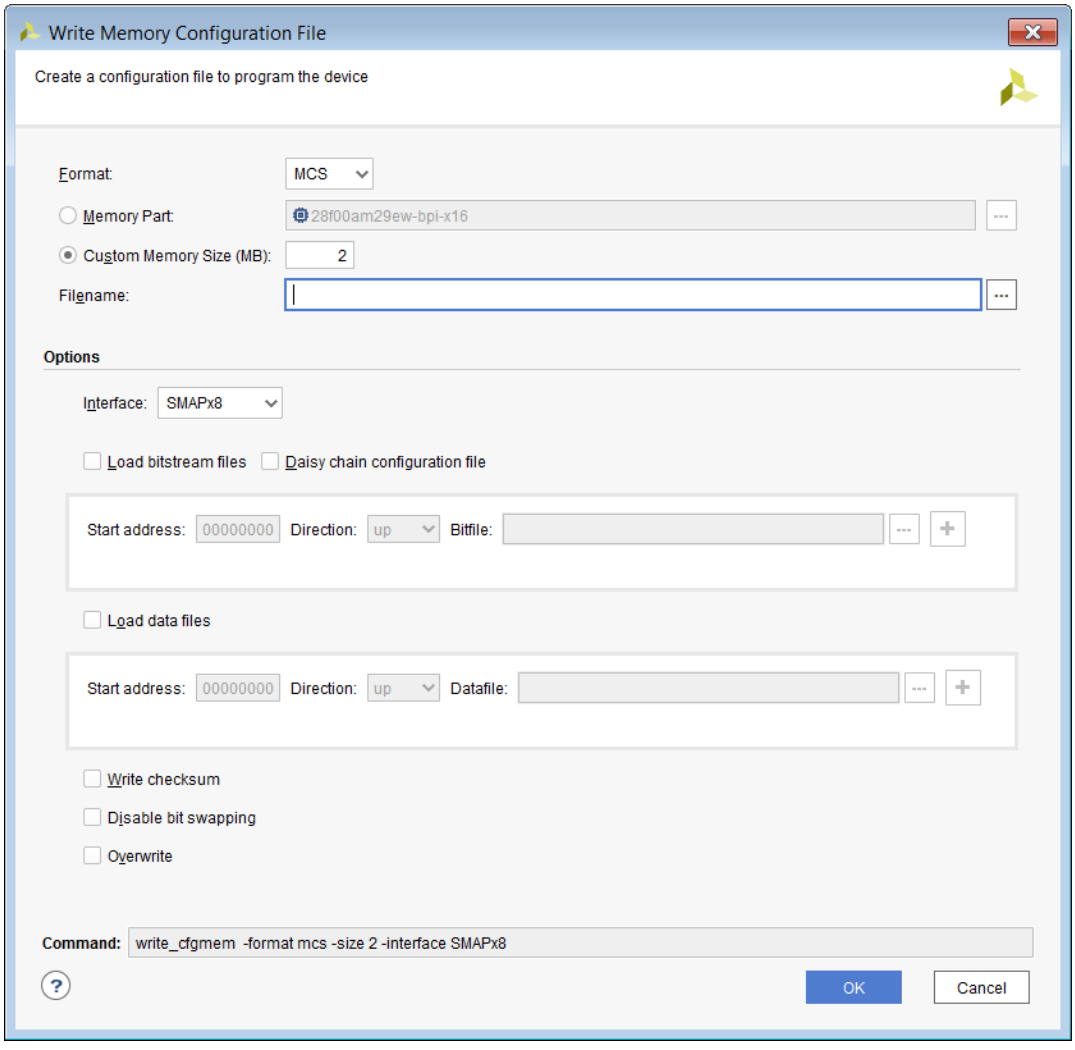
```
write_cfgmem -format mcs -interface bpix16 -size 128            \
        -loadbit "up 0 design1.bit up 0xFFFFF design2.bit"      \
        -file design1_design2.mcs
```

如需了解有关 write_cfgmem 命令的更多信息，请参阅《Vivado Design Suite Tcl 命令参考指南》(UG835)。

提示： 您可在 Vivado Lab Edition 中创建配置存储器文件。

您也可在 Vivado IDE 中创建配置存储器文件。单击“Tools” → “Generate Memory Configuration File”（工具 > 生成存储器配置文件）。这样会打开“Write Memory Configuration File”（编写存储器配置文件）对话框，如下所示：

图 23：**Write Memory Configuration File**



选择相应的格式和选项，然后单击“OK”以生成配置存储器文件。

为双 QSPI (x8) 器件创建配置存储器文件（适用于 FPGA 器件）

您可使用 `write_cfgmem` Tcl 命令来为双 QSPI (x8) 器件生成 `.mcs` 镜像。此命令会将配置数据自动拆分为 2 个独立的 `.mcs` 文件。

注释：为 SPIx8 生成 `.mcs` 时指定的大小即为这 2 个四通道闪存器件的总大小。

注释：`write_cfgmem` Tcl 命令在为双 QSPI (x8) 模式构建 `.mcs` 文件时会将起始地址一分为二。

write_cfgmem 使用示例

此示例演示了如何为多重启动设计生成 `.mcs` 文件，此设计在地址 0 加载“黄金镜像”比特流并在地址 0x0100_0000 加载多重启动比特流。

器件：2 个 256 Mib QSPI 闪存器件：`256 Mib = 32 MiB`

总存储空间大小：`2 * 32 MiB = 64 MiB`

加载地址：

黄金：0 * 2 = 0

多重启动：`0x0100_0000 * 2 = 0x0200_0000`

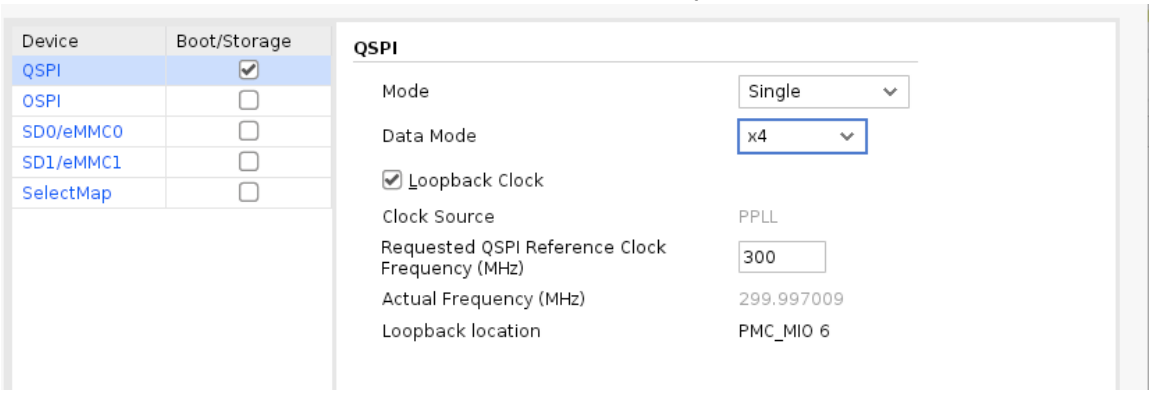
```
    write_cfgmem -format mcs -interface spix8 -size 32 \      -loadbit "up
0 ./design1_spix8.bit up 0x02000000 ./design2_spix8.bit" \      -file
design1_design2_spix8.mcs
```

为 Versal 器件创建初始化 PDI

AMD Versal™ 器件需要用户提供的器件镜像，用于在配置存储器烧录进程期间启动器件。此类器件镜像可以是设计 PDI，但在某些情况下（例如，需要采用不同的控制器选项时），对于此初始启动进程可采用不同的 PDI。

1. 创建以所需 Versal 器件为目标的新 AMD Vivado™ 工程，以供在存储器器件配置期间使用。
2. 创建完工程后，单击“IP INTEGRATOR → Create Block Design”（IP integrator > 创建块设计）以创建新的块设计。
3. 单击“+”图标将新的 IP 添加到 IP integrator 画布中，并搜索“Control, Interfaces & Processing System”。将“Control, Interfaces & Processing System IP”添加到 IP integrator 画布中。

4. 双击“Control, Interfaces & Processing System IP”，在 PS PMC 中配置选项。此时，应配置要在初始化 PDI 中使用的所需控制器选项。例如，要在配置存储器器件烧录期间使用 QSPI x4，应采用如下设置。



注释：初始化 PDI 中设置的选项将仅在配置存储器器件烧录步骤中使用，在烧录到配置存储器器件内的设计中，则不会沿用这些选项。

5. 在“Control, Interfaces & Processing System IP”中完成配置后，请运行块设计确认步骤，然后保存块设计。单击 Flow Navigator 中的“PROJECT MANAGER”（工程管理器）返回至“Project Navigator”（工程导航器），然后在“Sources”（源）窗格中右键单击新创建的块设计。选中“Create HDL Wrapper”（创建 HDL 封装文件），并选中“Let Vivado Manage wrapper and auto-update”（由 Vivado 管理封装文件并自动更新）。
6. 在 Flow Navigator 中，选择“PROGRAM AND DEBUG → Generate Device Image”（烧录和调试 > 生成器件镜像）以创建 PDI。

   注释：在此步骤期间创建的 PDI 会作为初始化 PDI，以供在配置存储器烧录进程中使用。
7. 至此初始化 PDI 已创建完成，应保存以供在本指南后续配置存储器烧录进程中使用。

连接到 Vivado 中的硬件目标

要连接到 AMD Vivado™ 中的硬件目标，请执行以下操作：

1. 要从闪存启动或配置 AMD 器件，请确保针对目标闪存类型选中模式管脚。请参阅对应您的目标器件的相应技术参考手册（对于 AMD Versal™ 器件，请参阅《Versal 自适应 SoC 技术参考手册》(AM011)）或配置用户指南。

   如需了解更多信息，请参阅目标器件的相应“配置用户指南”。
2. 遵循“器件烧录”中的步骤，连接至硬件目标。

重要提示！如果开发板已掉电或者电缆已断开连接，那么 Vivado IDE 会关闭硬件目标。同时还会取消 Vivado 主线程中的任意 Vivado 操作。

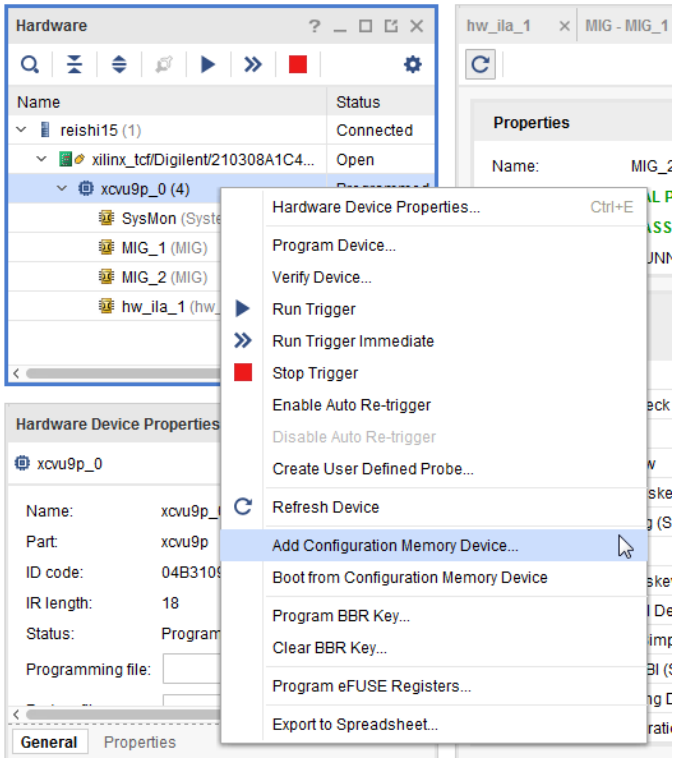
相关信息

器件烧录

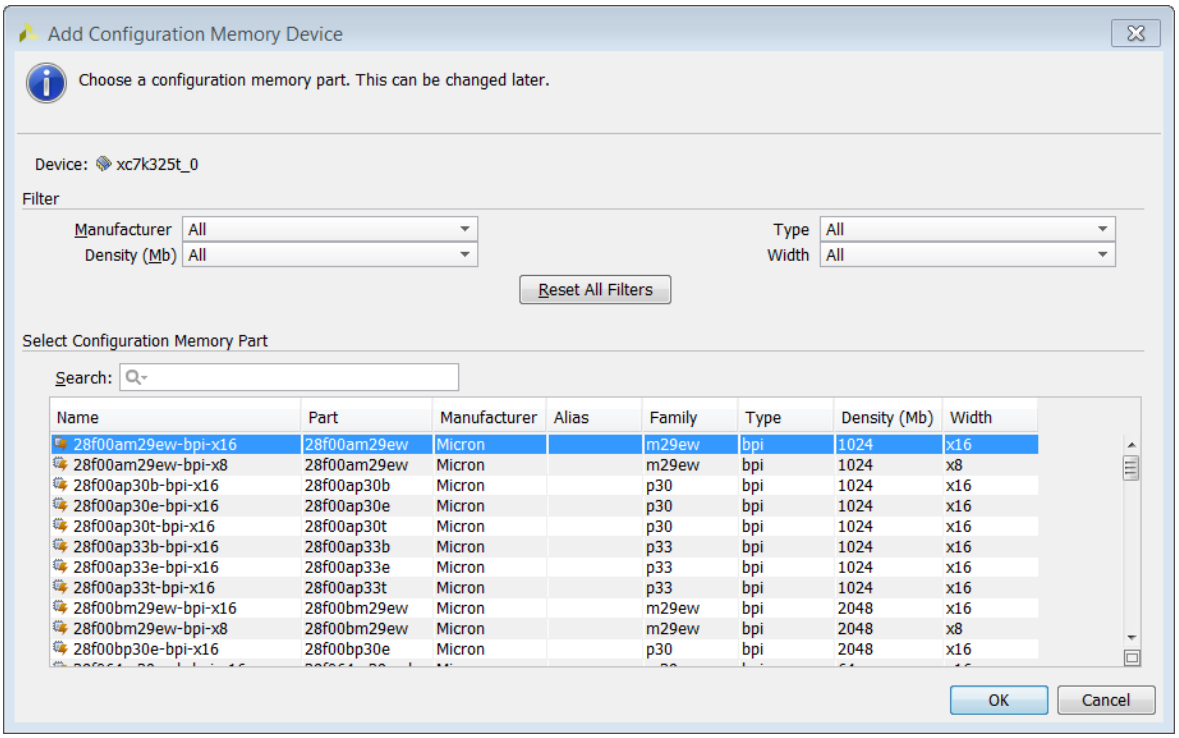
添加配置存储器器件

要将配置存储器器件添加至 AMD Vivado™ 器件烧录器中的硬件目标，请执行以下操作：

1. 按照所述连接至硬件目标后，右键单击硬件目标并单击“Add Configuration Memory Device”（添加配置存储器器件）以添加配置存储器器件。



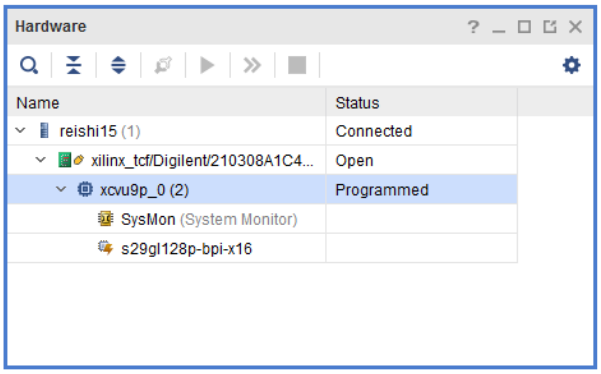
单击此菜单项即可打开“Add Configuration Memory Device”（添加配置存储器器件）对话框：



2. 选中相应的配置存储器部件，然后选择“OK”（确定）。

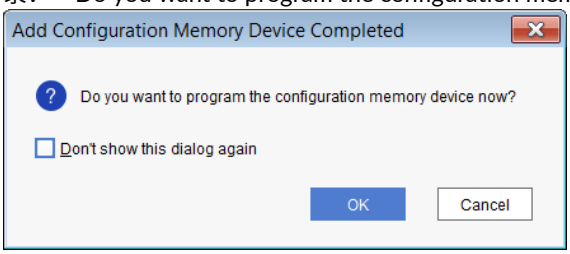
提示：选择“Manufacturer”（制造商）、“Density”（密度）或“Type”（类型）信息，并使用“Search”（搜索）字段缩小列表显示信息范围。

这样即可将配置存储器器件添加至硬件目标器件中。

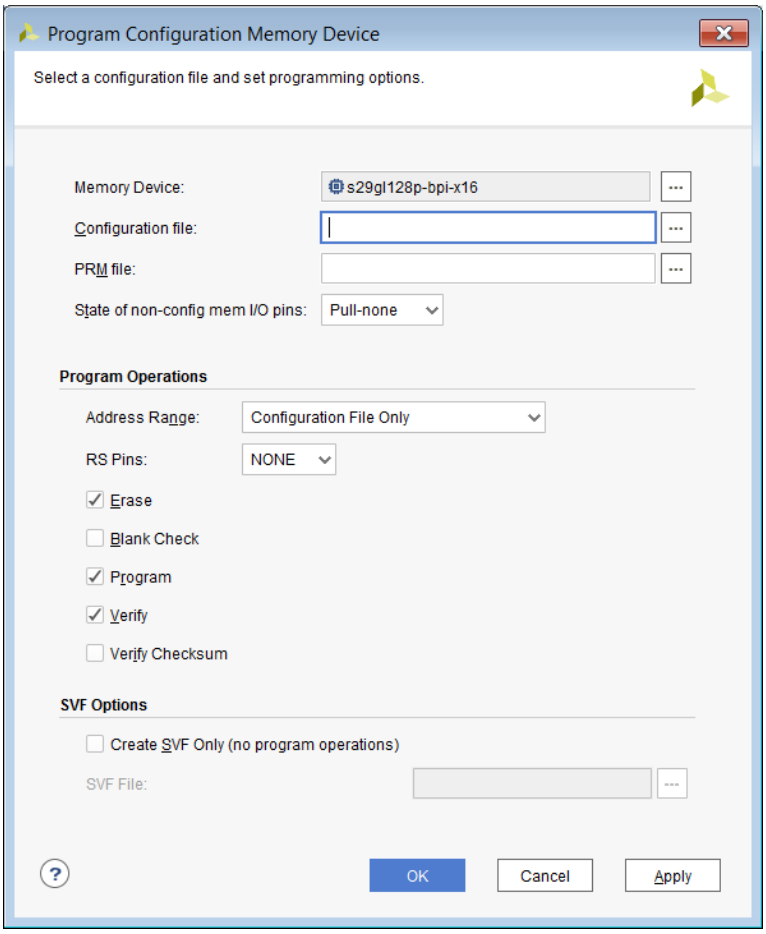


配置存储器器件烧录

1. 创建配置存储器器件后，AMD Vivado™ 器件烧录器会发出如下提示，询问您现在是否要对配置存储器器件进行烧录：“Do you want to program the configuration memory device now?”



单击“OK”即可打开“Program Configuration Memory Device”（配置存储器器件烧录）对话框。



2. 选中此对话框中的所有字段：

   - Configuration file（配置文件，`.mcs` 或 `.bin`）：指定用于烧录配置存储器器件的文件。存储器配置文件将使用 `write_cfgmem` Tcl 命令来创建。请参阅“创建配置存储器文件”，以获取更多信息。

   非配置存储器 I/O 管脚的状态：

   - Pull-none（无上拉下拉）：指定烧录到 FPGA 中的间接配置比特流将未使用的 I/O 管脚设置为“pull-none”。
   - Pull-up（上拉）：指定烧录到 FPGA 中的间接配置比特流将未使用的 I/O 管脚设置为“pull-up”。
   - Pull-down（下拉）：指定烧录到 FPGA 中的间接配置比特流将未使用的 I/O 管脚设置为“pull-down”。

   重要提示！确保非配置存储器 I/O 管脚的状态与 `write_bitstream` 属性中的设置相匹配。该属性默认值为 pull-down。

   在配置存储器器件上执行的烧录操作：

   - Address Range（地址范围）：指定要烧录的配置存储器器件的地址范围。可能的地址范围值如下。
   - Configuration File Only（仅限配置文件）：仅使用存储器配置文件所需的地址空间来执行擦除、空白检查、烧录和验证。
   - Entire Configuration Memory Device（整个配置存储器器件）：在整个器件上执行擦除、空白检查、烧录和验证。

- RS Pins（RS 管脚）：可选。Revision Select Pin Mapping（版本选择管脚映射）：仅限配合 BPI 配置存储器器件一起使用（其中闪存上的 2 个上位 FPGA 地址管脚绑定到 FPGA RS[1:0]）。启用该选项时，Vivado 会驱动 FPGA RS[1:0] 用于烧录。请参阅相应的 FPGA 配置用户指南，以了解应用的用法。
- Erase（擦除）：擦除配置存储器器件的内容。
- Blank Check（空白检查）：检查配置存储器器件，确保烧录前器件中不含数据。
- Program（烧录）：使用指定的配置文件（`.mcs` 或 `.bin`）对配置存储器器件进行烧录。
- Verify（验证）：烧录后，验证配置存储器器件的内容与配置文件（`.mcs` 或 `.bin`）相匹配。
- Verify Checksum（验证校验和）：确认配置存储器器件中已烧录的数据。该工具会基于配置存储器器件中已烧录的数据来计算校验和值，然后将其与 `.prm` 文件中指定的校验和值进行比较。

提示：用户可生成 cfgmem 文件并指定 `-checksum write_cftmem` 选项。此步骤会创建 `.prm` 文件，其中包含有关 cfgmem 输出文件的校验和信息。

- Create SVF Only（仅创建 SVF）：启用该选项即可支持以您指定的烧录操作来创建 SVF 文件。其他第三方工具可使用此 `.svf` 文件，在 Vivado 外部对配置存储器器件进行烧录。

重要提示！启用该选项后，Vivado 将生成含相关烧录选项的 SVF 文件。它不会实际对配置存储器器件执行烧录。

3. 单击“OK”以根据此对话框中的选择，在配置存储器器件上启动擦除、空白检查、烧录和验证操作。每项操作完成后，Vivado 都会通知您。

   注释：按下“Apply”（应用）将存储配置存储器设置，但不会对配置存储器器件进行烧录。如果您在按下“Apply”（应用）后按下“Cancel”（取消），那么配置存储器器件将进行置位，并且可稍后执行烧录。

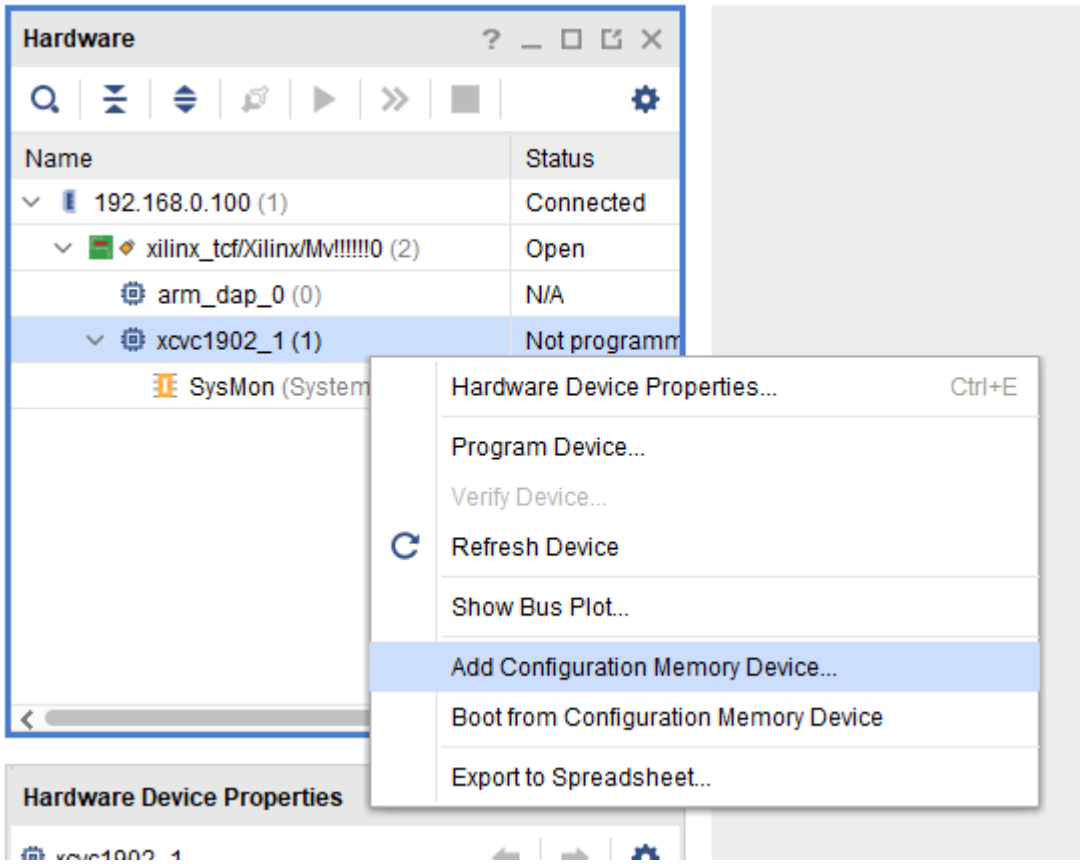
相关信息

创建配置存储器文件（适用于 FPGA 器件）

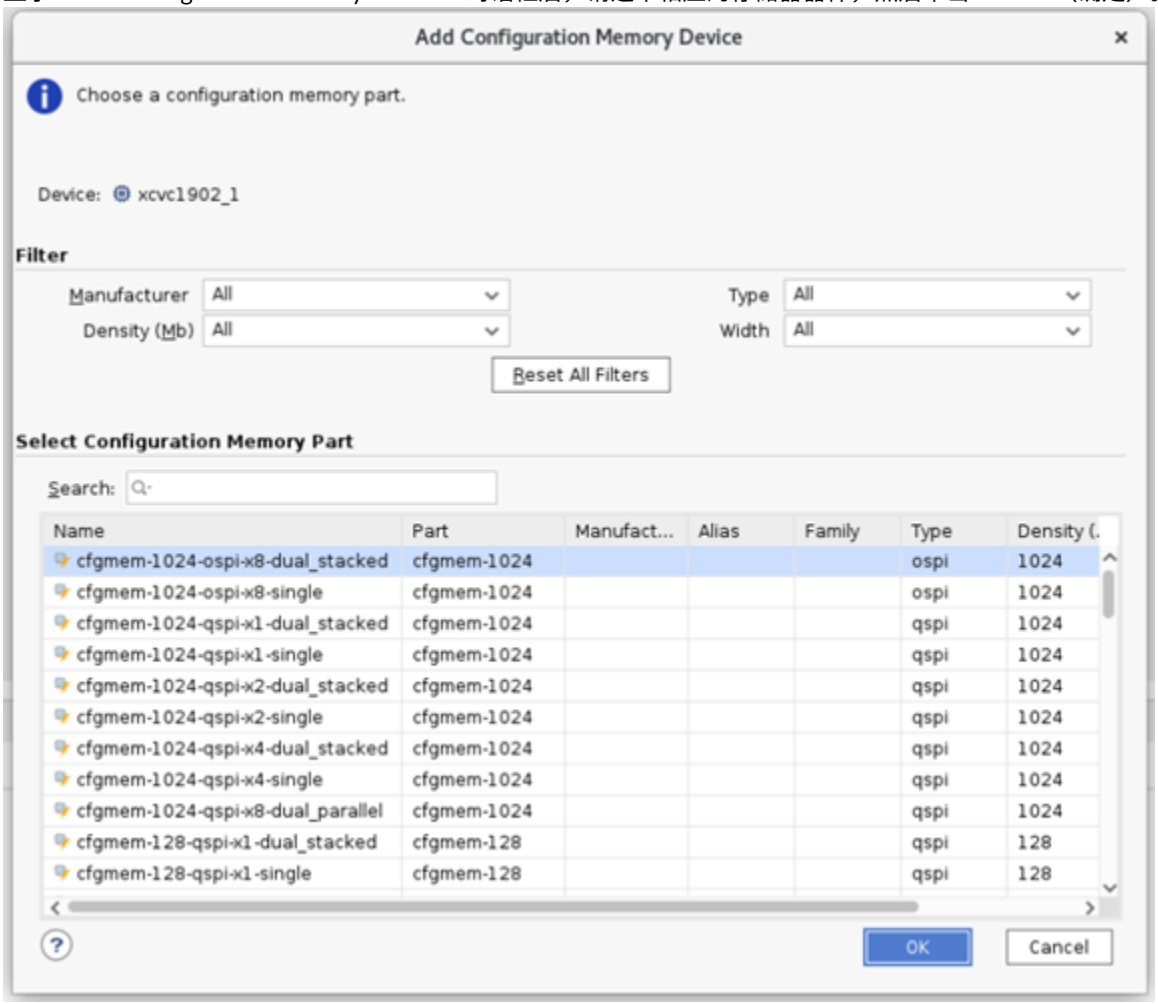
配置存储器器件烧录（Versal 器件）

创建初始 PDI 后，可使用以下步骤来对配置存储器器件进行烧录。

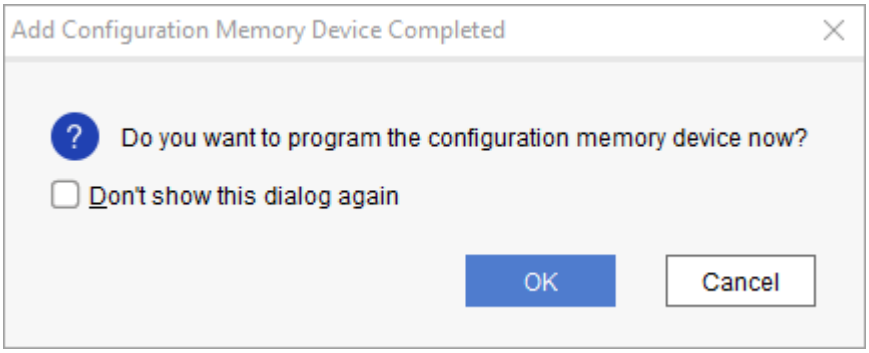
1. 如前述章节所述，启动 Vivado 硬件管理器，并连接到硬件目标。
2. 连接至硬件目标后，请右键单击硬件目标（如下所示）并单击“Add Configuration Memory Device”（添加配置存储器器件）来添加配置存储器器件。

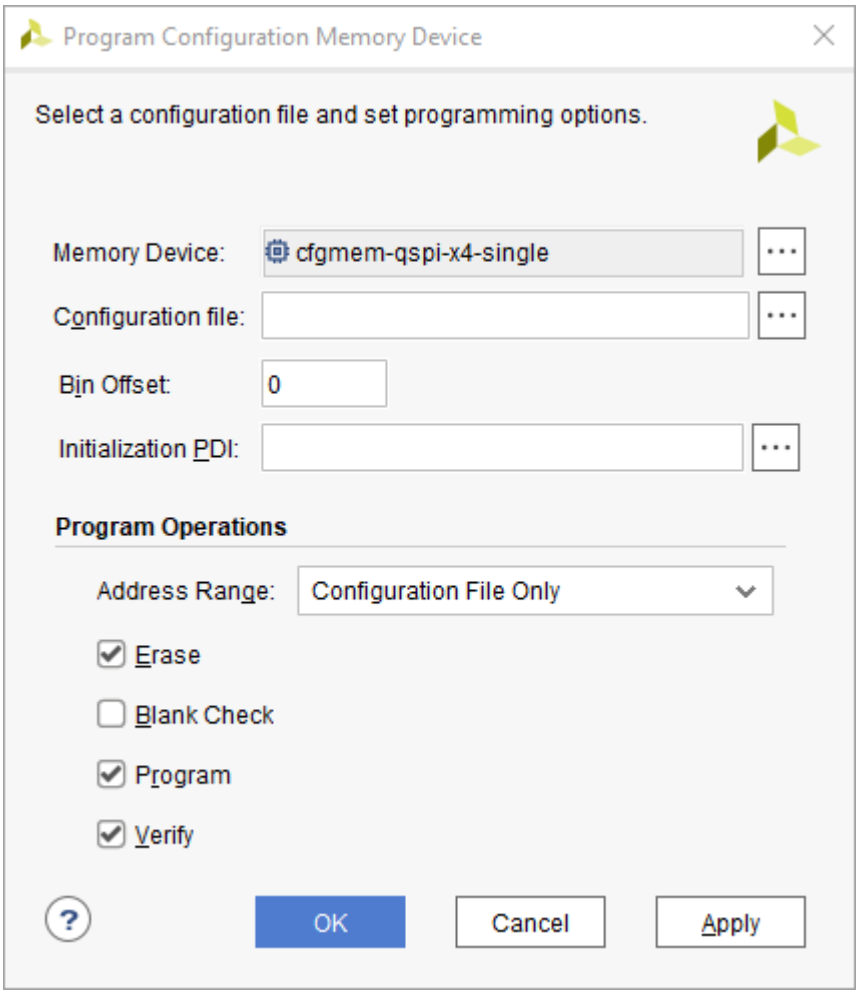

Hardware
Name
Status
192.168.0.100 (1)
Connected
xilinx_tcf/Xilinx/Mv!!!!!!0 (2)
Open
arm_dap_0 (0)
N/A
xcvc1902_1 (1)
Not programm
SysMon (System
Hardware Device Properties...
Ctrl+E
Program Device...
Verify Device...
Refresh Device
Show Bus Plot...
Add Configuration Memory Device...
Boot from Configuration Memory Device
Export to Spreadsheet...
Hardware Device Properties

3. 显示“Add Configuration Memory Device”对话框后，请选中相应的存储器器件，然后单击“OK”（确定）。



4. 这样即可将配置存储器器件添加至硬件目标中。要继续操作，Vivado 器件管理器将发出如下提示，询问您现在是否要对配置存储器器件进行烧录：“Do you want to program the configuration memory device now?”。单击“OK”（确定）以继续。





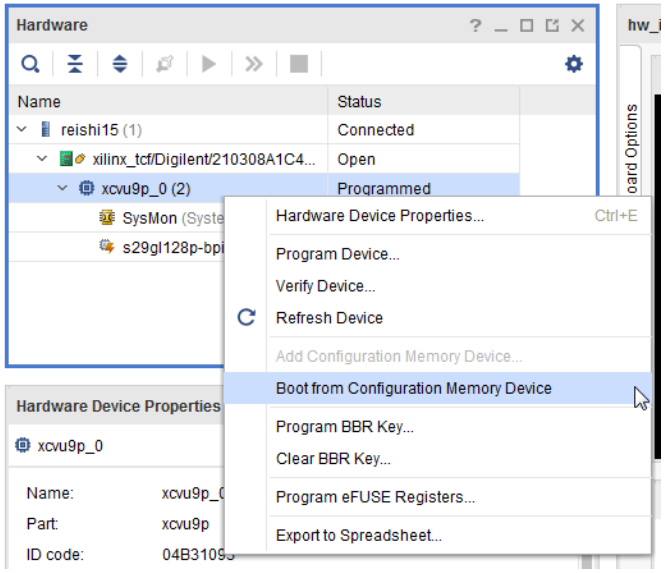
5. 在“Program Configuration Memory Device”（配置存储器器件烧录）窗口中，请对该对话框中的各字段进行相应设置：
   - “**Configuration file**”（配置文件）：指向要烧录到配置存储器中的 PDI 的路径。
   - “**Bin Offset**”（二进制偏移）：将器件镜像烧录到配置存储器中时要应用的偏移。
   - “**Initialization PDI**”（初始化 PDI）：指向用于在烧录前启动器件的 PDI 的路径。如果器件烧录所需的“CIPS IP”中所选的配置存储器控制器选项并无不同，那么此项可与先前指定的“**Configuration File**”相同。
   - “**Program Options**”（烧录选项）：选择“Configuration File Only”（仅限配置文件）将仅在要烧录的 PDI 所需的地址空间上执行所选的选项。选择“Entire Configuration Memory Device”（整个配置存储器器件）会在整个器件上执行所选的选项，包括擦除、空白检查、烧录和验证。
   - “**Erase**”（擦除）：擦除配置存储器器件的内容。
   - “**Blank Check**”（空白检查）：检查配置存储器器件，确保烧录前器件中不含数据。
   - “**Program**”（烧录）：使用所选器件镜像 (PDI) 对配置存储器器件进行烧录。
   - “**Verify**”（验证）：验证配置存储器器件内容与所选器件镜像 (PDI) 是否匹配。
6. 单击“OK”（确定）即可在配置存储器器件上启动选择操作。

   注释：按下“Apply”（应用）会存储配置存储器设置，但不会对配置存储器器件进行烧录。

启动 FPGA 器件

对配置存储器器件进行烧录后，您可发出软启动操作（即，JPROGRAM）以从所连接的配置存储器器件启动 FPGA 配置。要在目标 FPGA 上执行启动操作，请选中目标器件，右键单击并选择“Boot from Configuration Memory Device”（从配置存储器器件启动）。

图 24：**Boot from Configuration Memory Device**



重要提示！从配置存储器启动后，可能出现由于系统启动设置而导致调试核不立即显示的情况。AMD 建议您等待一段时间，具体时间可在 AMD Vivado™ 硬件管理器的 Tcl 控制台内使用 `boot_hw_device` Tcl 命令来指定，如下所示：

```
boot_hw_device after 1000 [refresh_hw_device]
```

其中，可指定的最大 `wait_on` 值为 1000。

在主模式下配置失败

注释：以下内容在 MPSoC 或 Versal 架构上不予支持。

当开发板处于“Master BPI”（主 BPI）模式或“Master SPI”（主 SPI）模式下且 JTAG 线缆连接至 Vivado 硬件管理器时，可能会发生配置失败。如果硬件管理器轮询和恢复功能导致主模式配置中断，那么在上电时可能会发生间歇性配置失败。为避免出现此问题，请在 Vivado 硬件管理器的 Tcl 控制台中设置以下参数，以确保不对配置状态寄存器进行更新：

```
set_param xicom.allow_cfgin_commands false
```

注释：此参数会影响整条链上的所有器件。

第 8 章

高级烧录功能

回读和验证

面向 FPGA 和 MPSoC 的比特流验证与回读

注释：以下内容不适用于 AMD Versal™ 架构。

AMD Vivado™ IDE 可对下载到 FPGA/MPSoC 中的配置数据（即，`.bit` 文件）进行验证和/或回读。使用 `write_bitstream` 生成 `.bit` 文件时，请使用 `-mask_file` 选项来创建对应的掩码 (`.msk`) 文件。在 Vivado IDE Tcl 控制台中运行 `write_bitstream -help` 即可获取有关比特流生成选项的详细信息。

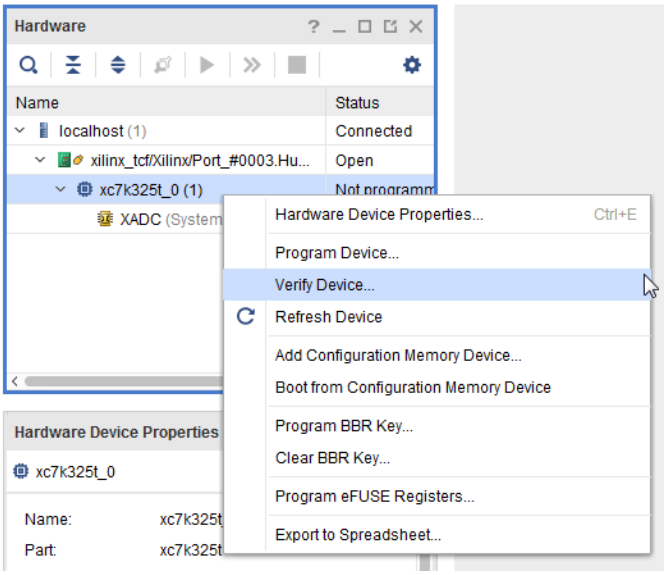
执行验证操作时，`verify_hw_devices` Tcl 命令会从 FPGA/MPSoC 回读数据，并使用 `.msk` 文件来判定哪些回读数据位可跳过以及哪些数据位用于与 `.bit` 文件中的对应位进行比较。

以下是比特流验证 Tcl 命令序列示例（`.bit` 和 `.msk` 文件是由上一次调用 `write_bitstream` 所生成的）：

```
create_hw_bitstream -hw_device [current_hw_device] \
          -mask kcu105_cnt_ila_uncmpr.msk  kcu105_cnt_ila_uncmpr.bit
verify_hw_devices [current_hw_device]
```

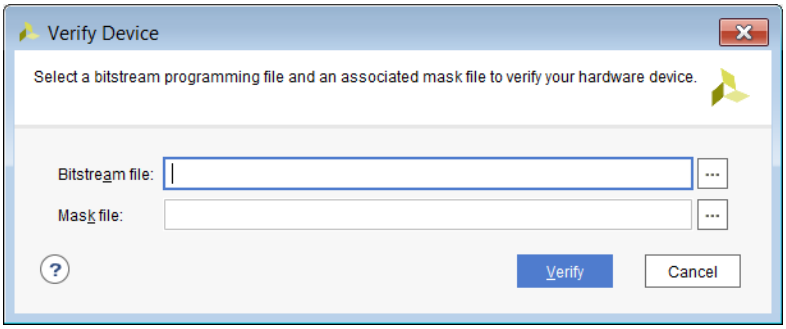
您可使用 Vivado 硬件管理器来验证配置数据。右键单击器件，然后单击“Verify Device”（验证器件），如下所示：

图 25：**验证器件选择**



这样会打开“Verify Device”（验证器件）对话框。

图 26： **“Verify Device”对话框**



您需要输入 BIT 文件和对应的掩码 (`.msk`) 文件。单击“Verify”（验证）以执行验证。

对以下至少一个选项使用 `readback_hw_device` Tcl 命令以回读 FPGA/MPSoC 配置数据：

- 要将回读数据保存为 ASCII 格式，请执行以下操作：

```
-readback_file <filename.rbd>
```

- 要将回读数据保存为二进制文件格式，请执行以下操作：

```
-bin_file <filename.bin>
```

示例：ASCII 格式和二进制文件格式的回读 FPGA/MPSoC 配置数据：

```
readback_hw_device [current_hw_device] \
     -readback_file  kcu105_cnt_ila_uncmpr_rb.rbd \
     -bin_file           kcu105_cnt_ila_uncmpr_rb.bin
```

1. 比特流和回读操作都是通过 Tcl 控制台完成的。
2. 验证和回读操作对于以加密比特流烧录的 FPGA 或 MPSoC 无效。加密比特流包含禁用回读的命令。要重新启用回读，可对器件的 PROG 管脚进行脉冲，或者将器件/开发板下电然后重新上电。
3. 使用 `readback_hw_device` 回读的数据仅包含配置数据（不包含配置命令）。

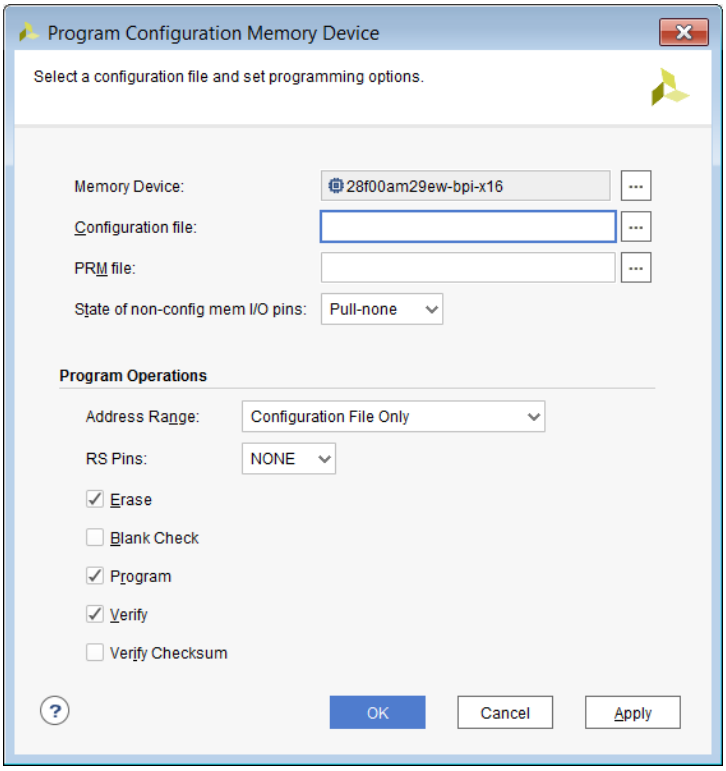
如需获取有关回读和掩码文件的更多信息，请参阅《UltraScale 架构配置用户指南》(UG570) 中的“验证回读数据”部分或者请参阅《7 系列 FPGA 配置用户指南》(UG470)。

FPGA 或 MPSoC 的配置和启动存储器验证与回读

您可将比特流文件 (`.bit`) 转换为 `.mcs` 或 `.bin` 文件，然后通过 write_cfgmem 命令将其烧录到配置存储器器件（例如，串行/SPI 或并行/BPI 闪存）中。欲知详情，请参阅《Vivado Design Suite Tcl 命令参考指南》(UG835)。

通过 AMD Vivado™ Design Suite 硬件管理器来验证配置存储器器件，如下图所示。

图 27：**配置存储器验证**



您还可以通过设置相应的 HW_CFGMEM 属性并调用 `program_hw_cfgmem` 来验证配置存储器器件，如以下代码所示：

```
set_property PROGRAM.ADDRESS_RANGE {use_file} [ get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]] set_property PROGRAM.FILES
[list "H:/projects/k7_led/k7_led_325t_afx_x16_33v.mcs" ] \ [ get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]] set_property
PROGRAM.BPI_RS_PINS {none} [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]] set_property PROGRAM.UNUSED_PIN_TERMINATION {pull-
none} [ get_property \ PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.BLANK_CHECK 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]] set_property PROGRAM.ERASE 0 [ get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]] set_property
PROGRAM.CFG_PROGRAM 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]] set_property PROGRAM.VERIFY 1 [ get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]] startgroup if {![string
equal [get_property PROGRAM.HW_CFGMEM_TYPE [lindex [get_hw_devices] 0]]
[get_property MEM_TYPE [get_property CFGMEM_PART [get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]]]] } \
{ create_hw_bitstream -hw_device [lindex [get_hw_devices] 0] [get_property
PROGRAM.HW_CFGMEM_BITFILE \ [ lindex [get_hw_devices] 0]];
program_hw_devices [lindex [get_hw_devices] 0]; }; program_hw_cfgmem -
hw_cfgmem [get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
endgroup
```

在 FPGA 器件上，您可通过 Vivado Design Suite Tcl 控制台使用以下命令序列来回读配置存储器的内容（在 MPSoC 上不受支持）：

```
readback_hw_cfgmem -file test.bin -hw_cfgmem \    [get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]]
```

注释：只能通过 Tcl 控制台来执行配置存储器回读操作。

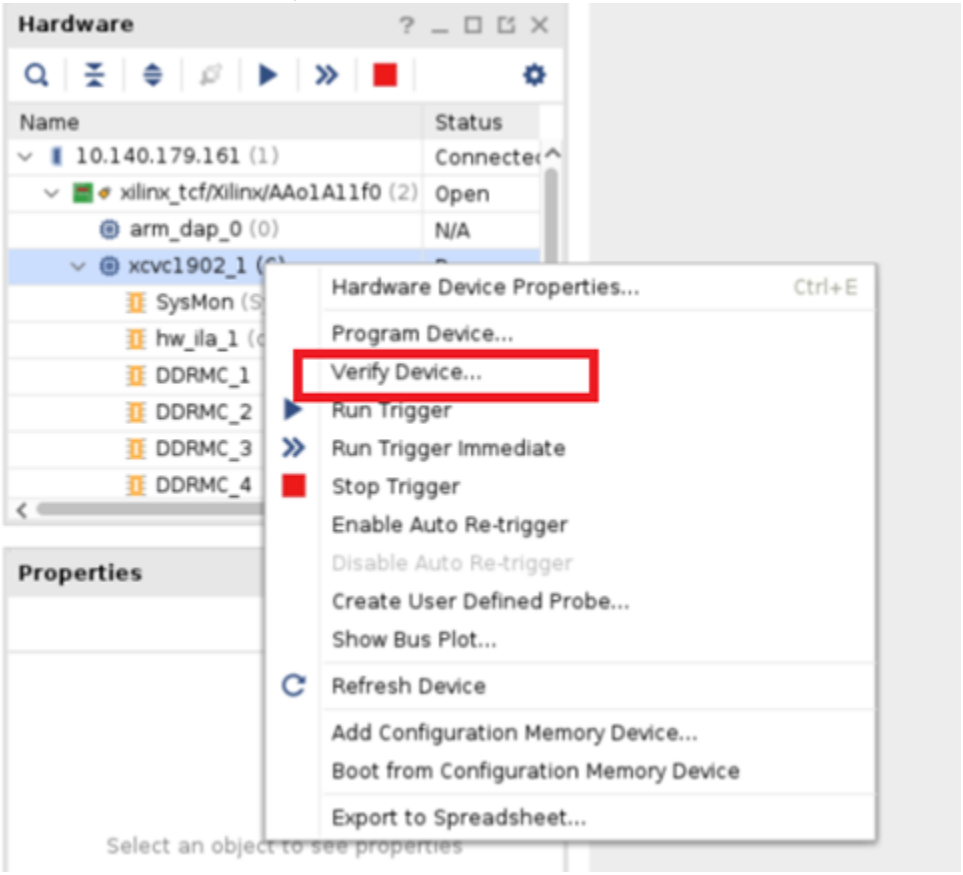
如需了解有关这些功能的更多信息，请参阅《UltraScale 架构配置用户指南》(UG570) 或《7 系列 FPGA 配置用户指南》(UG470)。

适用于 Versal 自适应 SoC 器件的 PDI 验证

下载到 Versal 自适应 SoC 器件的 PDI 可使用硬件管理器中的“verify Device ID”（验证器件 ID）选项或通过 `verify_hw_devices` Tcl 命令进行验证。`verify_hw_devices` 命令会将所选 PDI 的标识字段（镜像/节点 ID、唯一 ID、父唯一 ID 和功能 ID）与已烧录的器件上找到的标识字段进行比较。

要验证 PDI 与当前烧录到 Versal 器件中的镜像是否匹配，请执行以下操作：

1. 启动 Vivado 硬件管理器并连接到先前已烧录的目标。
2. 右键单击器件并选择“Verify Device”（验证器件），如下所示。

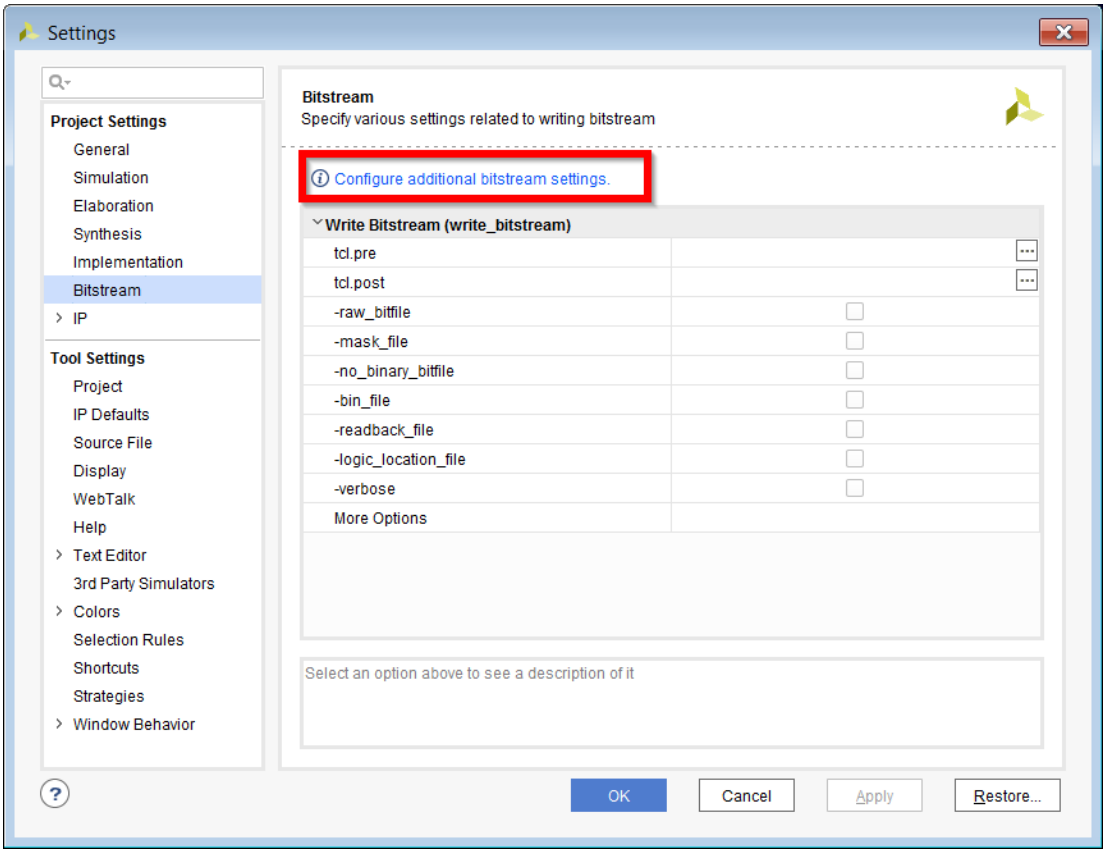


为 7 系列器件生成已加密文件和已经过身份验证的文件

注释：如需了解更多信息，请参阅《使用加密确保 7 系列 FPGA 比特流的安全》(XAPP1239)。

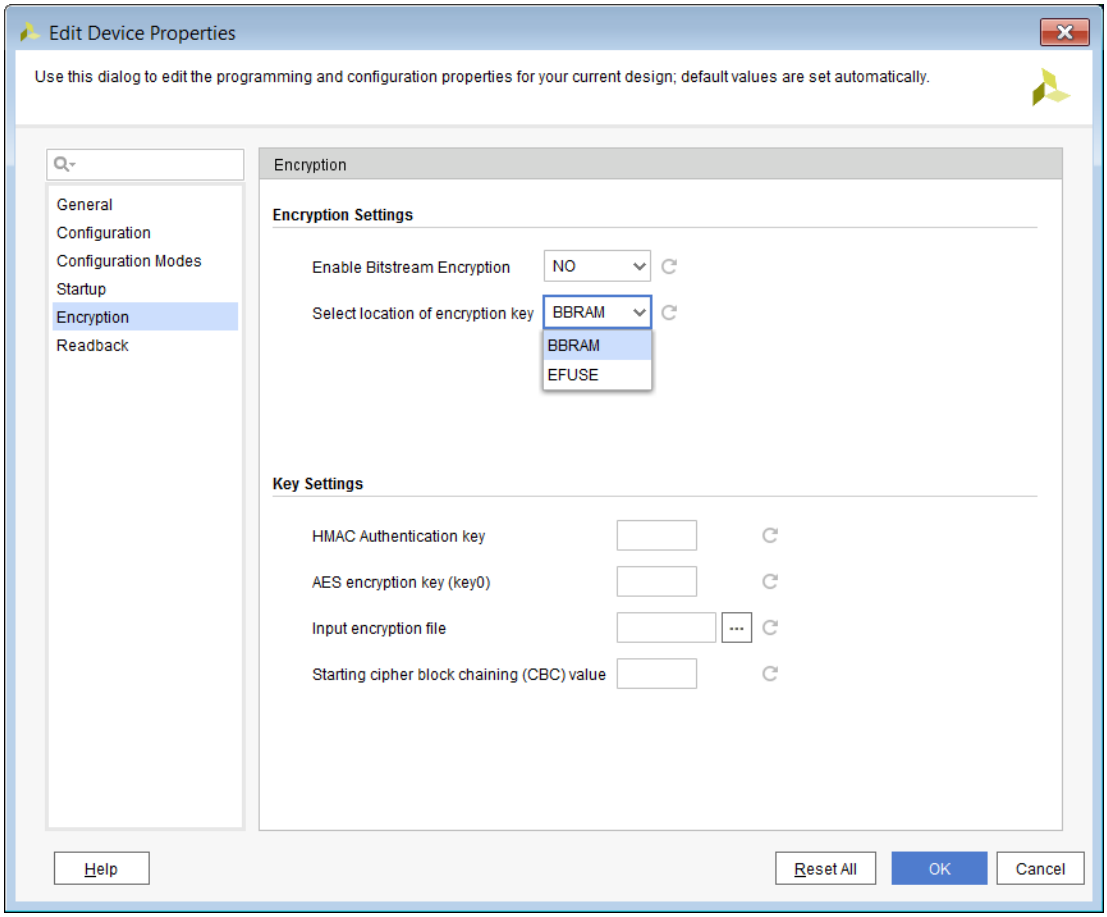
要生成加密比特流，请在 Vivado IDE 中打开已实现的设计。在主工具栏中，依次选择“Flow” → “Bitstream Settings”（流程 > 比特流设置），这样即可显示“Settings”（设置）对话框。在此对话框顶部，单击“Configure Additional Bitstream Settings”（配置其他比特流设置）。

图 28：**7 系列设置**



这样会显示“Edit Device Properties”（编辑器件属性）对话框。选择左侧窗格中的“Encryption”（加密）。

图 29：**7 系列配置加密设置**



在“Edit Device Properties”对话框中，指定“Encryption Settings”（加密设置）和“Key Settings”（密钥设置）：

- Encryption Settings（加密设置）
  - 将“Enable Bitstream Encryption”（启用比特流加密）设为“YES”（是）。
  - 将“Select location of encryption key”（选择加密密钥位置）设为“BBRAM”或“EFUSE”。
    - 密钥位置会嵌入加密比特流中。
    - 当加密比特流下载至器件后，它会指令 FPGA 使用已加载到 BBR 或 eFUSE 密钥寄存器中的密钥来对加密的比特流进行解密。
- Key Settings（密钥设置）
  - 指定 HMAC 身份验证密钥和密码分组链接 (CBC) 起始值。
    - 如果不指定这些值，则 Vivado 会为您生成随机值。
    - 这些值嵌入到加密比特流中，而无需烧录到 FPGA 中。

    注释：除非指定输入加密文件，否则这些值将存储在当前工程约束文件中。要避免将该值存储在约束文件中，请指定输入加密文件。
  - 指定加密比特流时要使用的“AES encryption key ”（AES 加密密钥）。您可使用最多 64 个十六进制字符来指定 256 位密钥。

- 此密钥将写入含 `.nky` 文件扩展名的文件中。将该密钥加载到 BBR 中时，或者将该密钥烧录到 eFUSE 密钥寄存器中时，请使用此文件。

  注释：除非指定输入加密文件，否则这些值将存储在当前工程约束文件中。要避免将该值存储在约束文件中，请指定输入加密文件。

- 指定输入加密文件。
  - 指定现有 `.nky` 文件即可获取加密密钥设置。该字段为可选字段，如果手动指定 AES、HMAC 和 CBC，则可省略该字段。

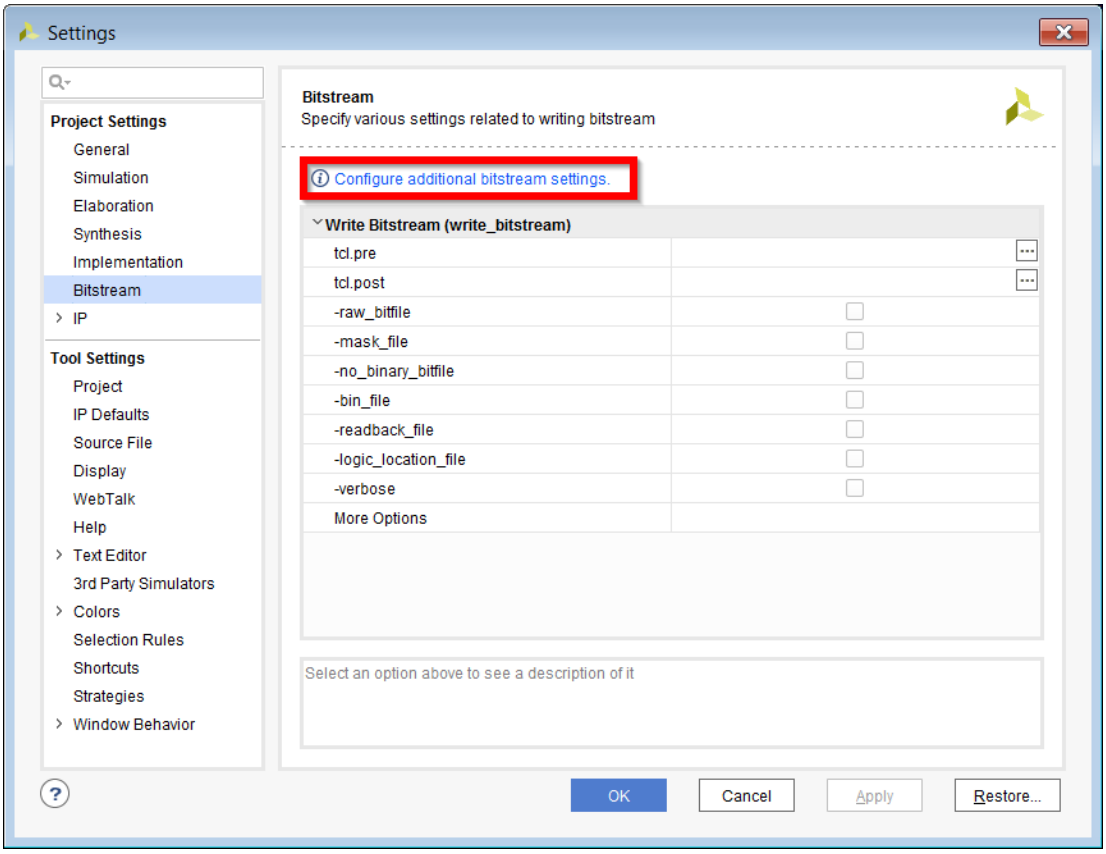
指定加密设置后，请单击“OK”（确定）以将这些设置应用于工程并重新生成比特流。完成 `write_bitstream` 操作后，您将获得一个烧录文件和一个 `.nky` 加密文件。

为 UltraScale 和 UltraScale+ 生成已加密文件和已经过身份验证的文件

注释：如需了解更多信息，包括有关 RSA 身份验证比特流支持的信息，请参阅《使用加密和身份验证确保 UltraScale/UltraScale+ FPGA 比特流的安全》(XAPP1267)。

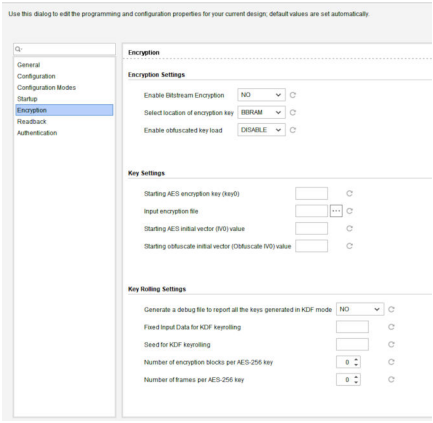
要生成加密比特流，请在 Vivado IDE 中打开已实现的设计。在主工具栏中，依次选择“Flow” → “Bitstream Settings”（流程 > 比特流设置），这样即可显示“Settings”（设置）对话框。在此对话框顶部，单击“Configure Additional Bitstream Settings”（配置其他比特流设置）。

图 30：**UltraScale 和 UltraScale+ 配置其他比特流设置**



这样会显示“Edit Device Properties”（编辑器件属性）对话框。选择左侧窗格中的“Encryption”（加密）。

图 31：**UltraScale 配置加密设置**



在“Edit Device Properties”对话框中，指定“Encryption Settings”（加密设置）和“Key Settings”（密钥设置），如下所示。

Encryption Settings（加密设置）

· 将“Enable Bitstream Encryption”（启用比特流加密）设为“YES”（是）。

- 将“Select location of encryption key”（选择加密密钥位置）设为“BBRAM”或“EFUSE”。
  - 密钥位置会嵌入加密比特流中。
  - 当加密比特流下载至器件后，它会指令 FPGA 使用已加载到 BBR 或 eFUSE 密钥寄存器中的密钥来对加密的比特流进行解密。

Key Settings（密钥设置）

- 指定加密比特流时要使用的“Starting AES encryption key (key0)”（起始 AES 加密密钥 (key0)）。您可使用最多 64 个十六进制字符来指定 256 位密钥。
  - 此密钥将写入含 `.nky` 文件扩展名的文件中。将该密钥加载到 BBR 中时，或者将该密钥烧录到 eFUSE 密钥寄存器中时，请使用此文件。

  注释：除非指定输入加密文件，否则该值将存储在当前工程约束文件中。要避免将该值存储在约束文件中，请指定输入加密文件。
- Specify Input encryption file（指定输入加密文件）：指定现有 `.nky` 文件即可获取加密密钥设置。该字段为可选字段，如果手动指定 AES、HMAC 和 CBC，则可省略该字段。
- 指定“Starting AES initial vector (IV0) value”（起始 AES 初始矢量 (IV0) 值）。选择对应第一个密钥的初始化矢量。

  注释：每个密钥都需要 1 个独立的初始化矢量值，该值可通过输入加密文件来提供。

  注释：该值将存储在当前工程约束文件中。要避免将该值存储在约束文件中，请指定输入加密文件。
- 指定“Starting obfuscate initial vector (Obfuscate IV0)”（起始模糊初始矢量（模糊 IV0））值。选择对应模糊密钥的初始化矢量。

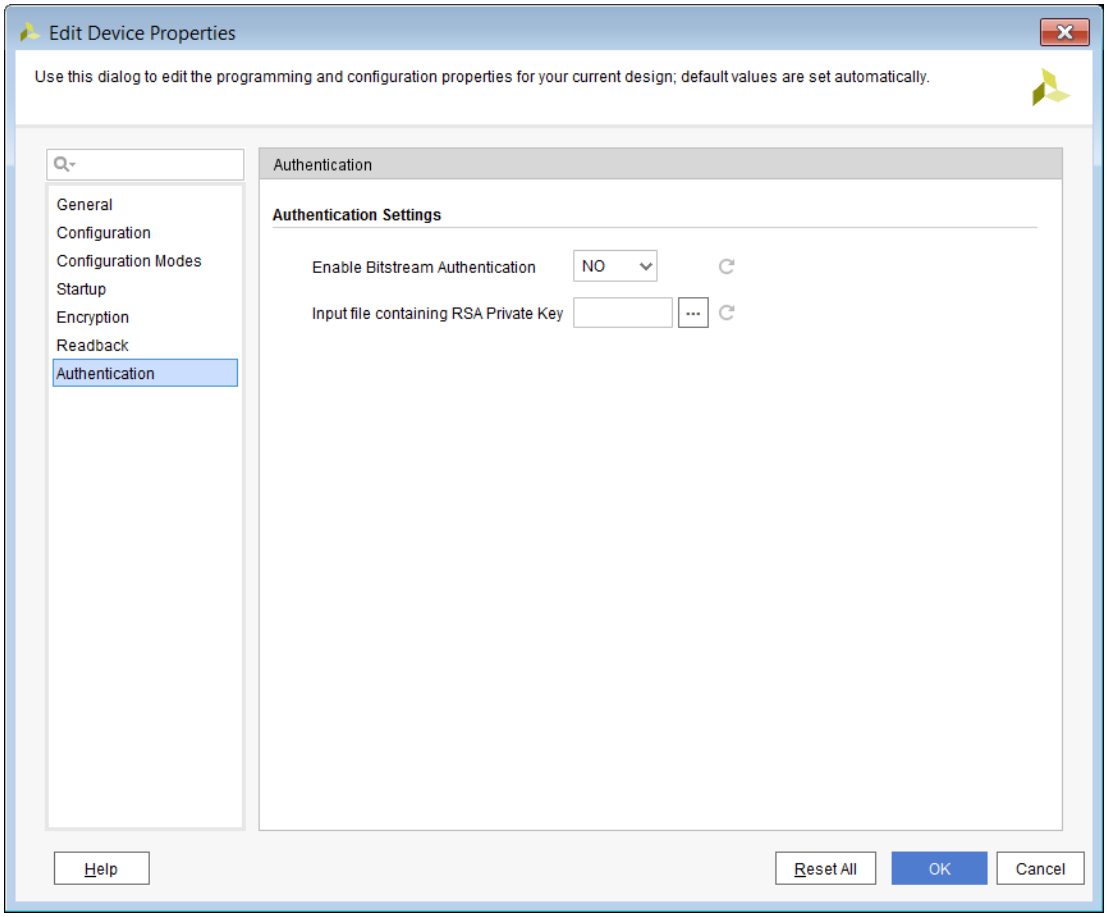
  注释：该值将存储在当前工程约束文件中。要避免将该值存储在约束文件中，请指定输入加密文件。

Key Rolling Settings（密钥滚动设置）

- 指定是否应生成调试文件以报告 KDF 模式下生成的所有密钥。
- 指定“Fixed Input Data for KDF key rolling”（用于 KDF 密钥滚动的固定输入数据）。该值为可选 60 字节固定输入值，并指定为 120 位十六进制值。此 60 字节输入搭配 4 字节计数器即可通过 RAND_bytes 充当 KDF 虚拟随机固定输入值的 64 字节输入。
- 指定“Seed for KDF Keyrolling”（用于 KDF 密钥滚动的种子值）。该值为 KDF 的可选 32 字节种子值，指定为 64 位十六进制值。
- 指定“Number of encryption blocks per key”（每个密钥的加密块数）和“Number of frames per AES-256 key”（每个 AES-256 密钥的帧数）。加密块数和帧数用于指定使用不同密钥将每个比特流分成多少节。

要执行身份验证设置，请选择左侧窗格中的“Authentication”（身份验证）

图 32：**Edit Device Properties - Authentication**



在“Edit Device Properties - Authentication”（编辑器件属性 - 身份验证）对话框中，指定如下加密和密钥设置：

Authentication Settings

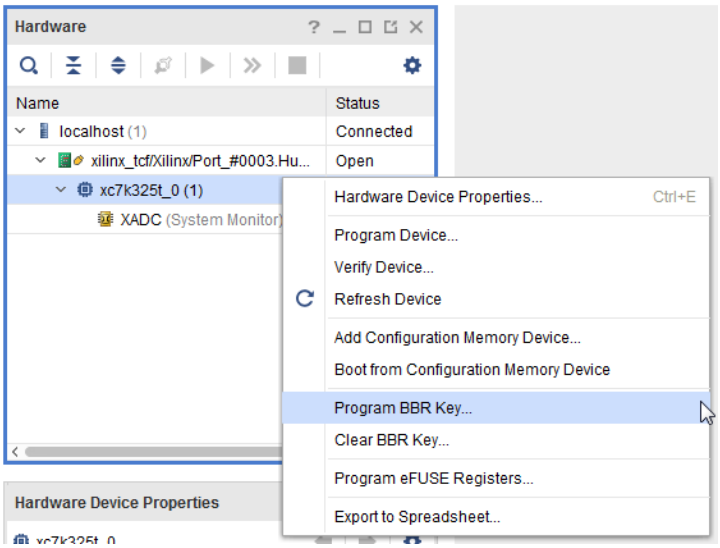
- 将“Enable Bitstream Authentication”（启用比特流身份验证）设置为“YES”（是）。
- 指定“Input file containing RSA Private Key”（包含 RSA 专用密钥的输入文件）。
- 指定加密和身份验证设置后提供 RSA 专用密钥，单击“OK”（确定），这样即可将这些设置应用于工程。重新运行实现，并重新生成比特流文件。成功完成 `write_bitstream` 操作后，生成的 `.nky` 加密密钥文件将显示在加密比特流文件所在目录中。

您可通过使用 256 位高级加密标准 (AES) 密钥加密比特流，下载这些比特流，并仅在经授权的 FPGA 上运行来保护比特流中的 IP。具体方法是将 256 位密钥烧录到经授权的 FPGA 的 BBR 寄存器中，然后再下载加密比特流。

面向 7 系列器件的 AES 密钥烧录

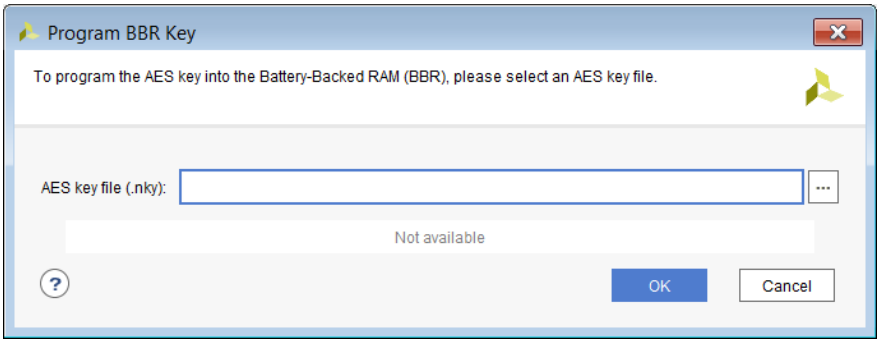
要将 AES 密钥烧录到 BBR 中，请在“Hardware”（硬件）窗口中右键单击 FPGA，然后单击“Program BBR Key”（BBR 密钥烧录）。

图 33：**BBR 密钥烧录**



在“Program BBR Key”对话框中，输入文件名或者浏览至目标文件以指定 AES 密钥文件 (`.nky`)。指定有效的 `.nky` 文件后，就会自动填充 AES 密钥字段。单击“OK”使硬件管理器将密钥烧录或加载到 BBR 中。

图 34：**BBR 密钥烧录 - 7 系列**



完成密钥烧录后，请使用满足下列条件的加密比特流对 FPGA 进行烧录：

- 加密时使用的是加载到 BBR 中的 AES 密钥。
- 已选中 BBRAM 作为指定的加密密钥位置。

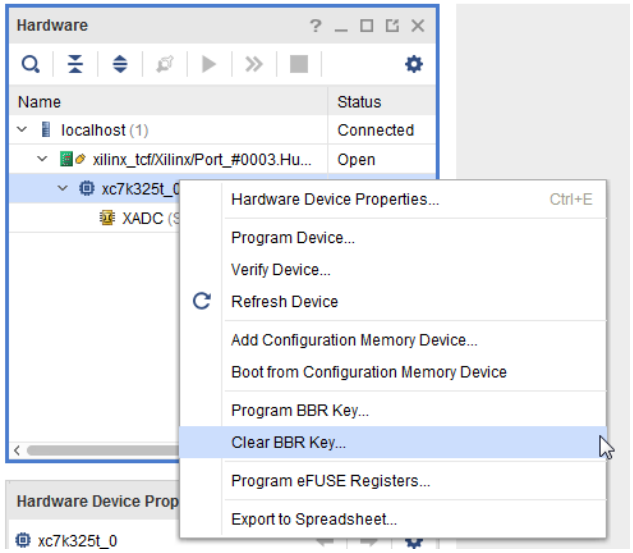
为 7 系列器件清空 AES 密钥

要手动清空 AES 密钥，请断开 Vbatt 管脚的连接，将开发板下电，然后重新上电。

注释：当板或 FPGA 上电时，按下或脉冲 PROG 管脚并不会清空 BBR 寄存器。

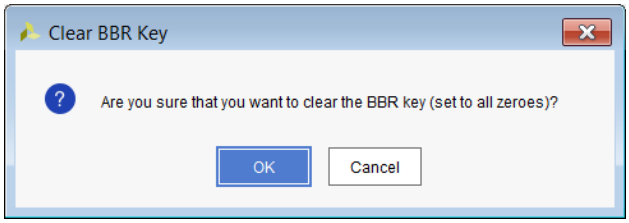
您也可选择在 Vivado IDE 的“Hardware”（硬件）窗口中右键单击 FPGA 并选择“Clear BBR Key”（清空 BBR 密钥）来清空 AES 密钥

图 35：**为 7 系列器件清空 AES 密钥**



当“Clear BBR Key”对话框出现时，单击“OK”以从器件中清空密钥。

图 36：**“Clear BBR Key”对话框**

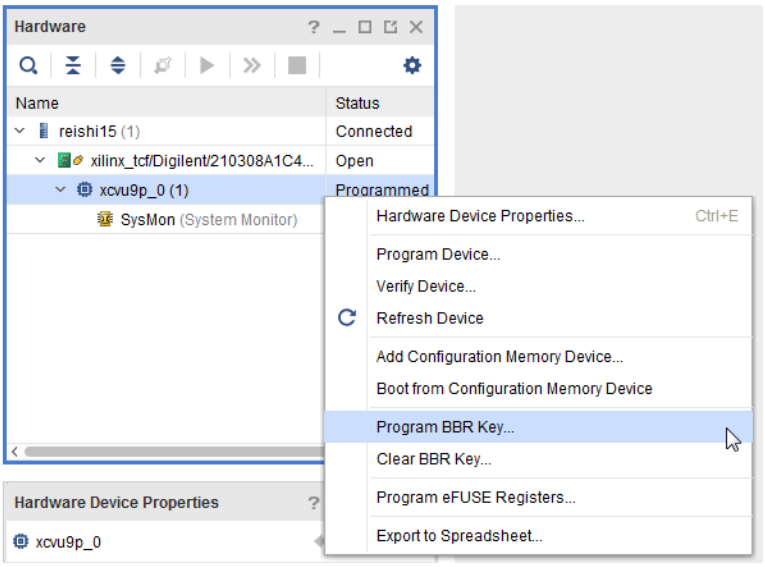


重要提示！在 BBR 密钥上执行 `verify_hw_devices` 时，将显示错误。要验证 BBR 密钥，用户应使用包含密钥的比特流对 FPGA 进行烧录以便对其进行测试。Vivado 不支持采用任何 BBR 烧录后验证选项来对烧录的 BBR 密钥进行验证。

为 UltraScale 和 UltraScale+ 器件执行 AES 密钥烧录

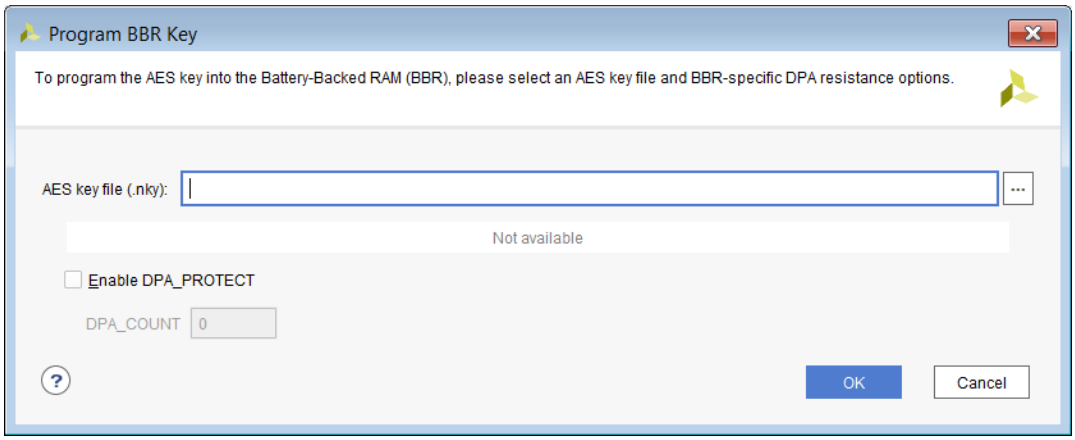
要将 AES 密钥烧录到 BBR 中，请在“Hardware”（硬件）窗口中右键单击 FPGA 并选中“Program BBR Key”（BBR 密钥烧录）。

图 37：**从“Hardware”窗口执行 BBR 密钥烧录**



这样会打开“Program BBR Key”对话框。

图 38：**BBR 密钥烧录 - UltraScale 和 UltraScale+ 器件**



在“Program BBR Key”对话框中，请指定 AES 密钥文件 (`.nky`) 并选中“Enable DPA PROTECT”（启用 DPA 保护），如下所示。

1. 输入文件名或者浏览目标文件以指定 AES 密钥文件 (.nky)。指定有效的 `.nky` 文件后，就会自动填充 AES 密钥字段。
2. 选中“Enable DPA PROTECT”复选框。
3. 指定 DPA_COUNT 值。启用此项时，有效值范围为 1 - 256。

   注释：如需获取有关 BBR AES 密钥和 DPA_PROTECT 功能的更多详细信息，请参阅《UltraScale 架构配置用户指南》(UG570)。
4. 单击“OK”，使硬件管理器将密钥烧录或加载到 BBR 中。
5. 完成密钥烧录后，请使用满足下列条件的加密比特流对 FPGA 进行烧录：此加密比特流使用的是加载到 BBR 中的 AES 密钥，并选中 BBRAM 作为指定的加密密钥位置。

重要提示！对于 UltraScale 器件，如果您在将密钥烧录到 BBR 寄存器之前已下载了加密比特流（使用 BBR 作为密钥源），那么 FPGA 会锁定，并且您无法加载 BBR 密钥。您仍可下载未加密的比特流，但无法下载加密比特流，因为 FPGA 会阻止您将密钥下载到 BBR 中。您必须将开发板下电，然后重新上电，才能解锁此 UltraScale 器件，然后才能重新加载 BBR 密钥。

重要提示！在 BBR 密钥上执行 `verify_hw_devices` 时，将显示错误。要验证 BBR 密钥，用户应使用包含密钥的比特流对 FPGA 进行烧录以便对其进行测试。Vivado 不支持采用任何 BBR 烧录后验证选项来对烧录的 BBR 密钥进行验证。

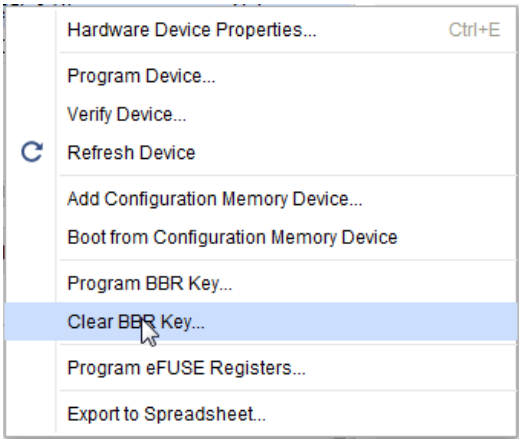
为 UltraScale 和 UltraScale+ 器件清空 AES 密钥

要手动清空 AES 密钥，请断开 Vbatt 管脚的连接，将开发板下电，然后重新上电。

注释：当板或 FPGA 上电时，按下或脉冲 PROG 管脚并不会清空 BBR 寄存器。

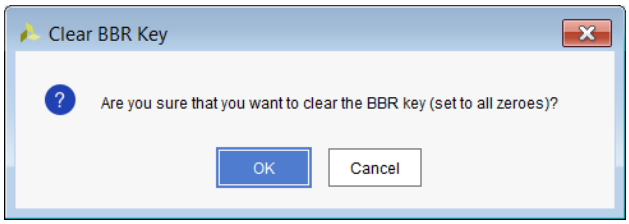
您也可选择在 Vivado IDE 的“Hardware”（硬件）窗口中右键单击 FPGA 并选择“Clear BBR Key”（清空 BBR 密钥）来清空 AES 密钥。

图 39：**为 UltraScale 和 UltraScale+ 器件清空 AES 密钥**



当“Clear BBR Key”对话框出现时，单击“OK”以从器件中清空密钥。

图 40：**“Clear BBR Key”对话框**



eFUSE 寄存器访问和烧录

注释：在 MPSoC 和 Versal 器件上不支持以下 eFUSE 访问和烧录方法。

7 系列、UltraScale 和 UltraScale+ 器件具有一次性可烧录位（称为 eFUSE 位），这些位用于执行特定功能。不同 eFUSE 位类型如下所述：

- FUSE_DNA - 存储唯一器件标识位（不可烧录）。
- FUSE_USER - 存储 32 位用户定义的代码。
- FUSE_KEY - 存储密钥以供 AES 比特流解密器使用。
- FUSE_CNTL - 控制密钥使用和对 eFUSE 寄存器的读写访问权。
- FUSE_SEC - 控制 UltraScale 器件和 UltraScale+ 器件中的特殊器件安全性设置。

重要提示！ eFUSE 寄存器位烧录只是一次性操作。eFUSE 寄存器位一经烧录（从未烧录状态 0 转换为已烧录状态 1），就无法复位为 0 和/或重新烧录。在对任意 eFUSE 寄存器进行烧录前，应谨慎核查设置。

注意！ 如有任何 eFUSE 寄存器位先前已烧录（从未烧录状态 0 转换为已烧录状态 1），那么尝试对其再次进行烧录时，Vivado 硬件管理器会发出严重警告，以指出部分位已烧录。但即使出现此警告，先前操作期间尚未烧录的后续 eFUSE 寄存器位（处于未烧录状态 0）仍会继续进行烧录。

重要提示！ AMD 建议首先对 FUSE_USER、FUSE_KEY 和 FUSE_RSA 寄存器进行烧录，然后重新运行“eFUSE Programming”Wizard（eFUSE 烧录向导），对 FUSE_SEC 位进行烧录以控制 FPGA 安全性设置，最后对 FUSE_CNTL 位进行烧录以控制对这些 eFUSE 位执行的读写访问。

针对 eFUSE 烧录的电缆支持

支持 eFUSE 烧录的兼容 JTAG 下载线缆和器件列表如下：

- AMD SmartLynq 数据线缆 (HW-SMARTLYNQ-G/DLC20)
- AMD 平台电缆 USB II (DLC10)
- Digilent JTAG-HS1
- Digilent JTAG-HS2
- Digilent JTAG-HS3

适用于 7 系列器件的 eFUSE 寄存器访问和烧录

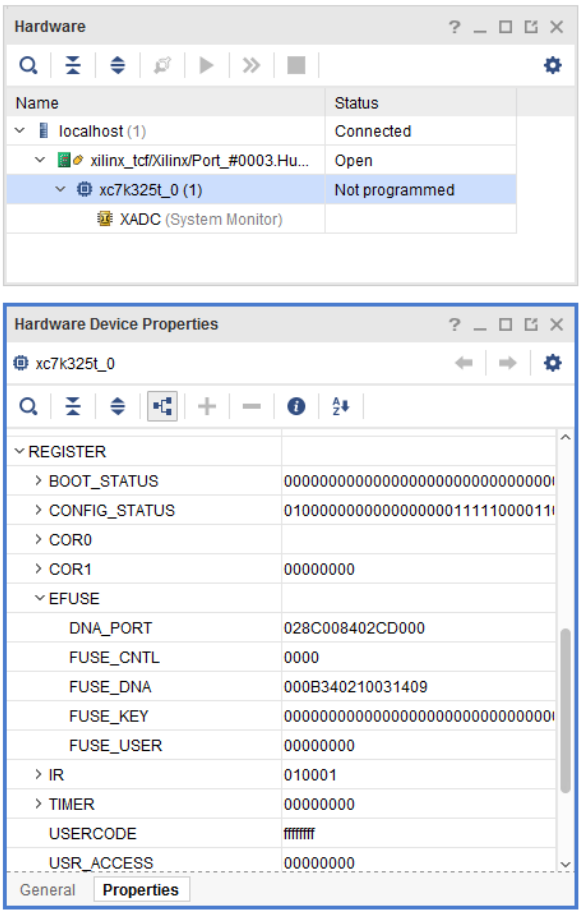
FUSE_DNA：唯一的器件 DNA

每个 7 系列器件都有唯一的器件 ID，称为器件 DNA，且 AMD 已将此 DNA 烧录到器件中。7 系列器件具有 64 位 DNA。您可在 Vivado Design Suite Tcl 控制台中运行以下 Tcl 命令来读取这些位：

```
get_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_DNA
```

您也可以在 Vivado Design Suite 的“Hardware Device Properties”（硬件器件属性）窗口中查看 eFUSE 寄存器来访问器件 DNA，如下图所示。

图 41：**eFUSE DNA**

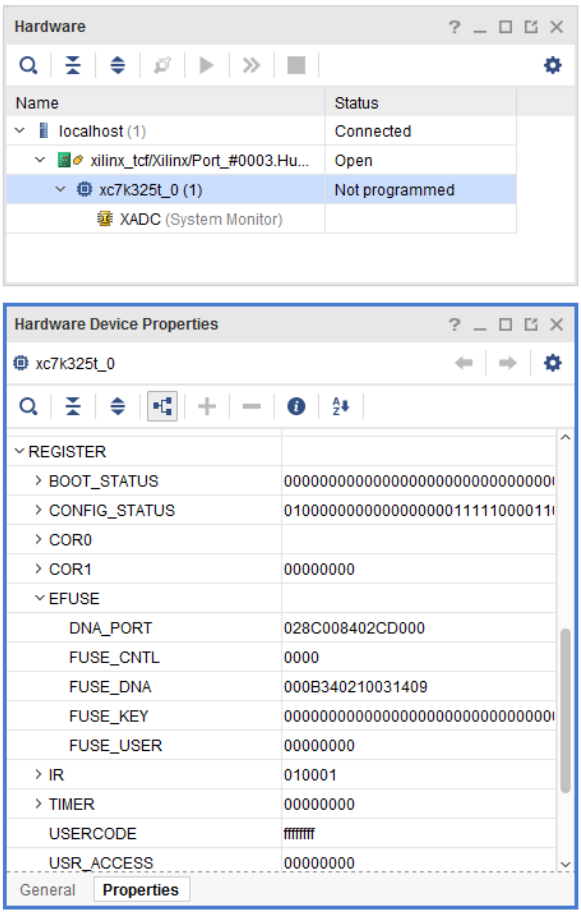


如需了解有关这些功能的更多信息，请参阅《7 系列 FPGA 配置用户指南》(UG470)。

eFUSE 寄存器烧录

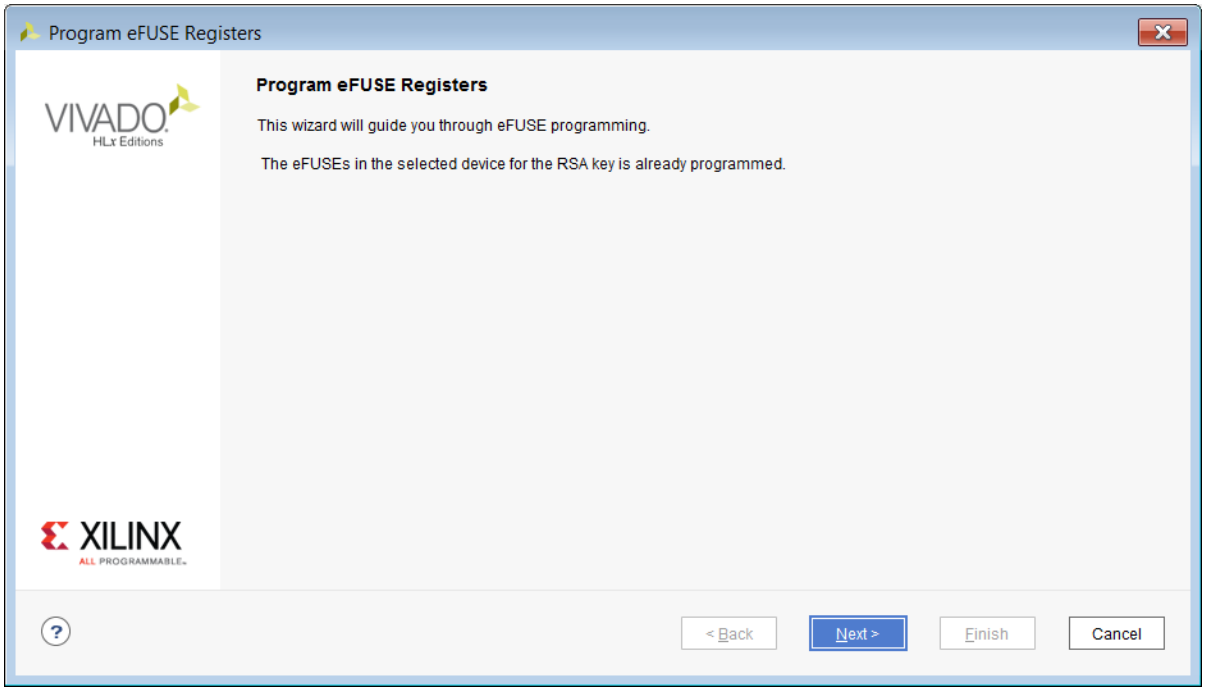
要执行 eFUSE 寄存器烧录，请在“Hardware”（硬件）窗口中右键单击 FPGA，然后单击“Program eFUSE Registers”（eFUSE 寄存器烧录）。

图 42：**选择“Program eFUSE Registers”**



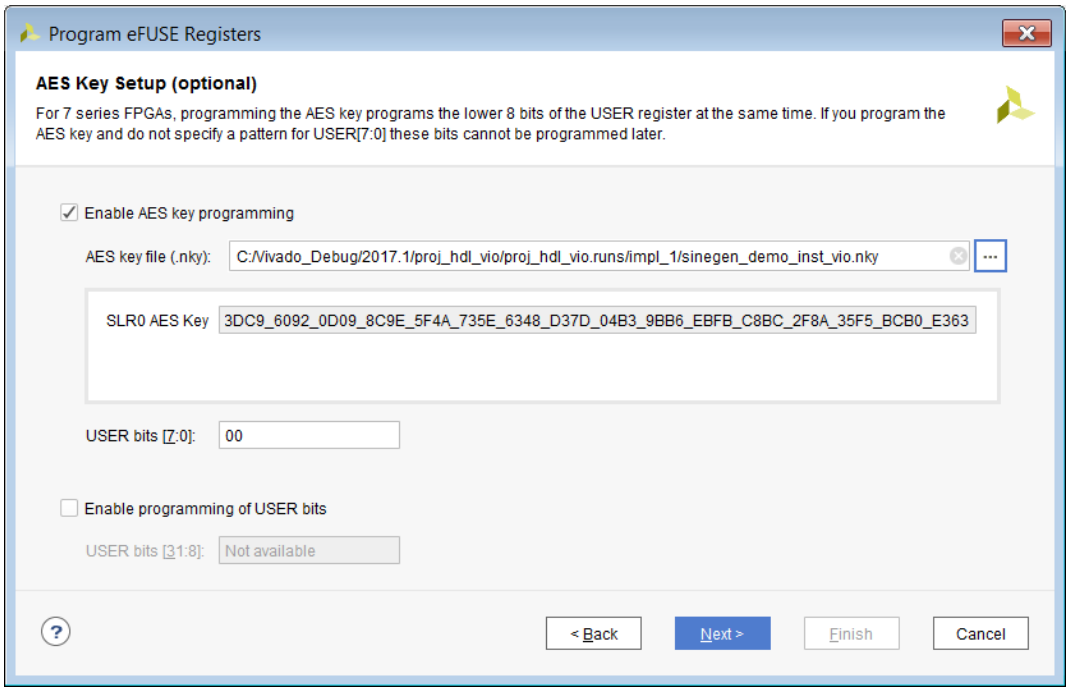
这样会显示如下图所示的“Program eFUSE Registers”Wizard（eFUSE 寄存器烧录向导），此向导将指导您为 eFUSE 寄存器设置各选项。

图 43：**“Program eFUSE Registers”Wizard**



在“AES Key Setup”（AES 密钥设置）窗格中，指定以下设置：

图 44：**eFUSE AES 密钥设置**



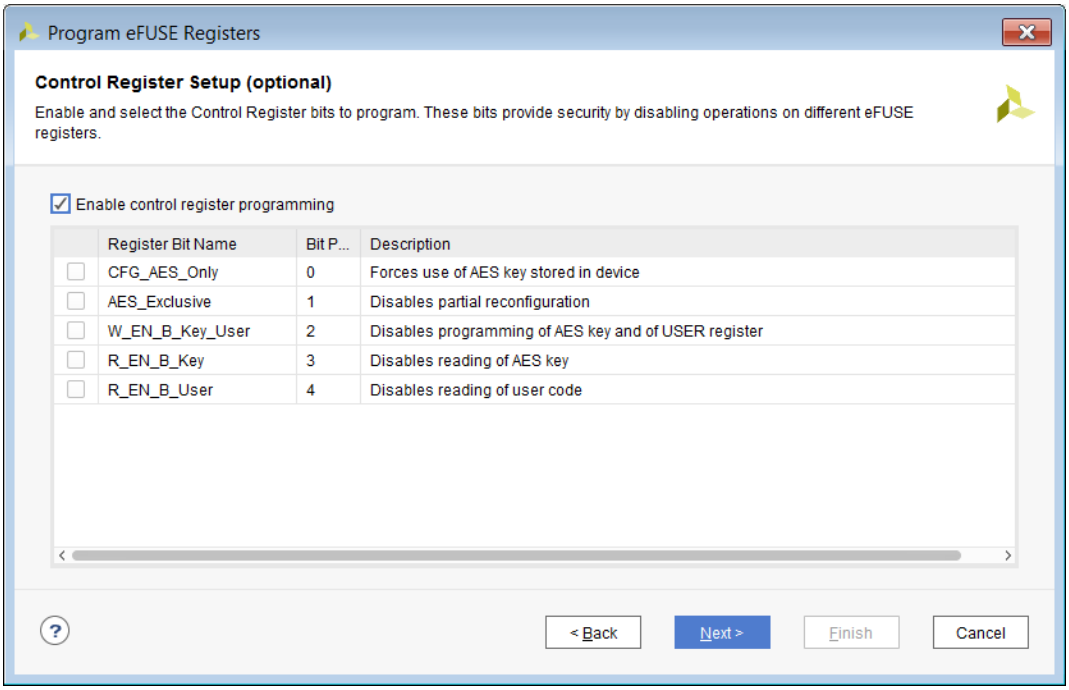
- “AES Key file”（AES 密钥文件）

- 输入文件名或者浏览至目标文件以指定 AES 密钥文件 (`.nky`)。指定有效的 `.nky` 文件后，就会自动填充 AES 密钥字段。

- “USER bits [7:0] and USER bits [31:8]” （USER 位 [7:0] 和 USER 位 [31:8]）
  - 所提供的 USER eFUSE 位可支持用户通过烧录来获取其自己的特殊 32 位模式。下 8 位 FUSE_USER 与 256 位高级加密标准 (AES) 密钥同时进行烧录。上 24 位用户位可与 AES 密钥并发烧录，或者也可稍后再烧录。

在“Control Register Settings” （控制寄存器设置）窗格中，指定以下设置：

图 45：**控制寄存器设置**

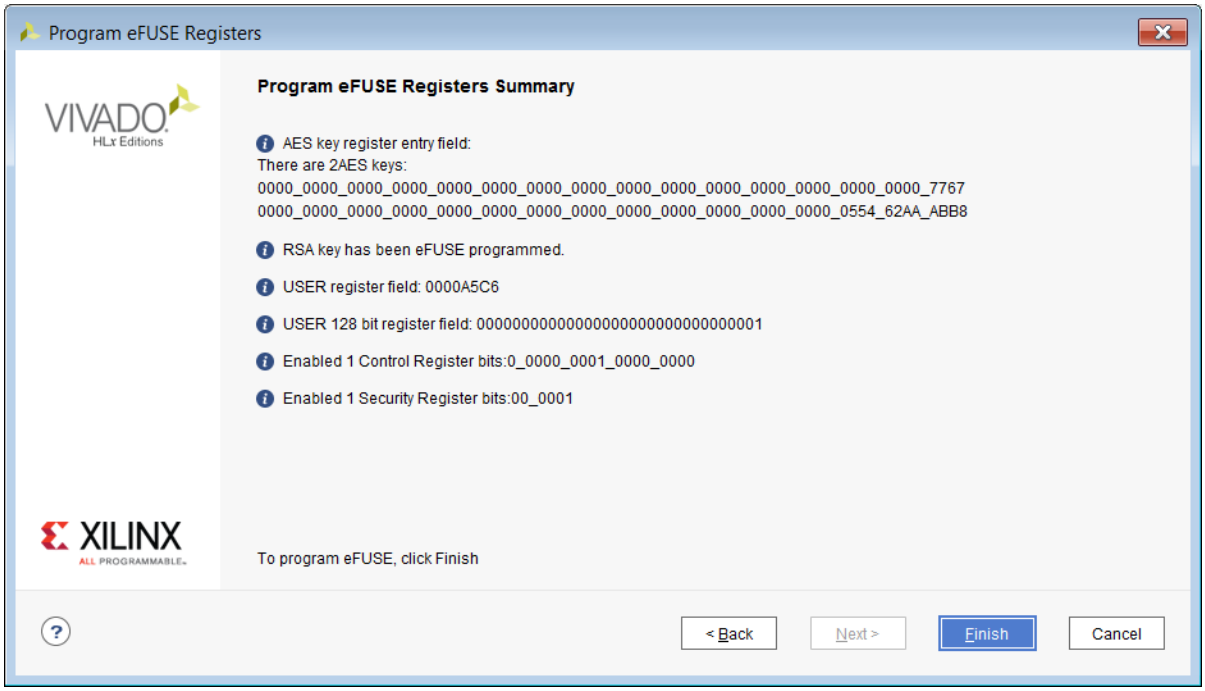


- CFG_AES_Only：设置此项即可强制使用存储的 AES 密钥。
- AES_Exclusive：设置此项即可禁用部分重配置。
- W_EN_B_Key_User：设置此项即可禁用 AES 密钥和用户寄存器的烧录操作。
- R_EN_B_Key：设置此项即可禁用 AES 密钥的读取操作。
- R_EN_B_User：设置此项即可禁用用户代码的读取操作。
- W_EN_B_Cntl：设置此项即可禁用此控制寄存器的烧录操作。

如需了解有关这些功能的更多信息，请参阅《7 系列 FPGA 配置用户指南》(UG470)。

请复查“Program eFUSE Registers Summary” （eFUSE 寄存器烧录汇总）页面中的 eFUSE 设置。

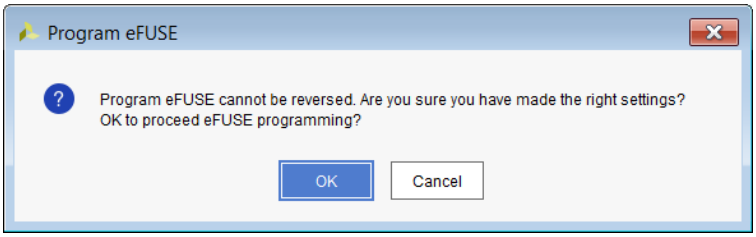
图 46：**Program eFUSE Registers Summary**



在此窗格中可显示“Program eFUSE Registers”Wizard（eFUSE 寄存器烧录向导）面板中设置的所有位。在此窗格中，您可查看各个位的设置，以便复查特定烧录设置。请仔细复查此汇总页面，以确保要烧录的每个位都已正确设置。

单击“Finish”（完成）即可显示“Program eFUSE”（eFUSE 烧录）确认对话框：

图 47：**“Program eFUSE”确认**



单击“OK”（确定）以对指定的 FUSE 位进行烧录。

强制执行 eFUSE 烧录

要对位于寄存器中任意位置的任何位（无论该位先前是否已烧录）强制执行置位，可将 `-force_efuse` 选项设置为 `program_hw_devices`。使用该选项时，将仅执行基本寄存器边界检查。

适用于 UltraScale 和 UltraScale+ 器件的 eFUSE 寄存器访问和烧录

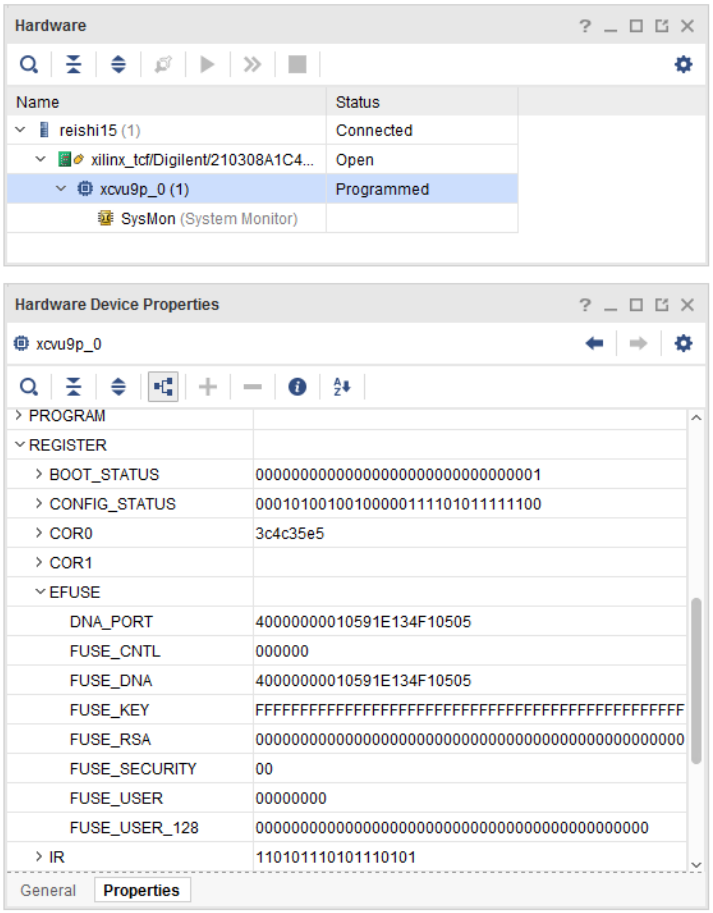
FUSE_DNA：唯一的器件 DNA

每个 UltraScale 器件都有唯一的器件 ID，称为器件 DNA，且 AMD 已将此 DNA 烧录到器件中。用户无法对 FUSE_DNA 进行烧录。UltraScale 器件具有 96 位 DNA。您可在 Vivado Design Suite Tcl 控制台中运行以下 Tcl 命令以读取 FUSE_DNA：

```
get_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_DNA
```

您也可以在 Vivado Design Suite 的“Hardware Device Properties”（硬件器件属性）窗口中通过查看 eFUSE 寄存器来访问器件 DNA，如下图所示：

图 48：**UltraScale 和 UltraScale+ 器件的 eFUSE DNA**

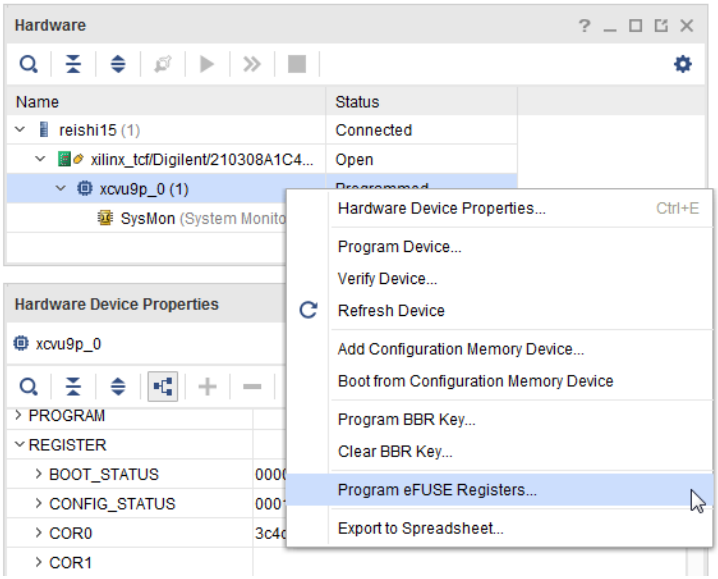


如需了解有关这些功能的更多信息，请参阅《UltraScale 架构配置用户指南》(UG570)。

eFUSE 寄存器烧录

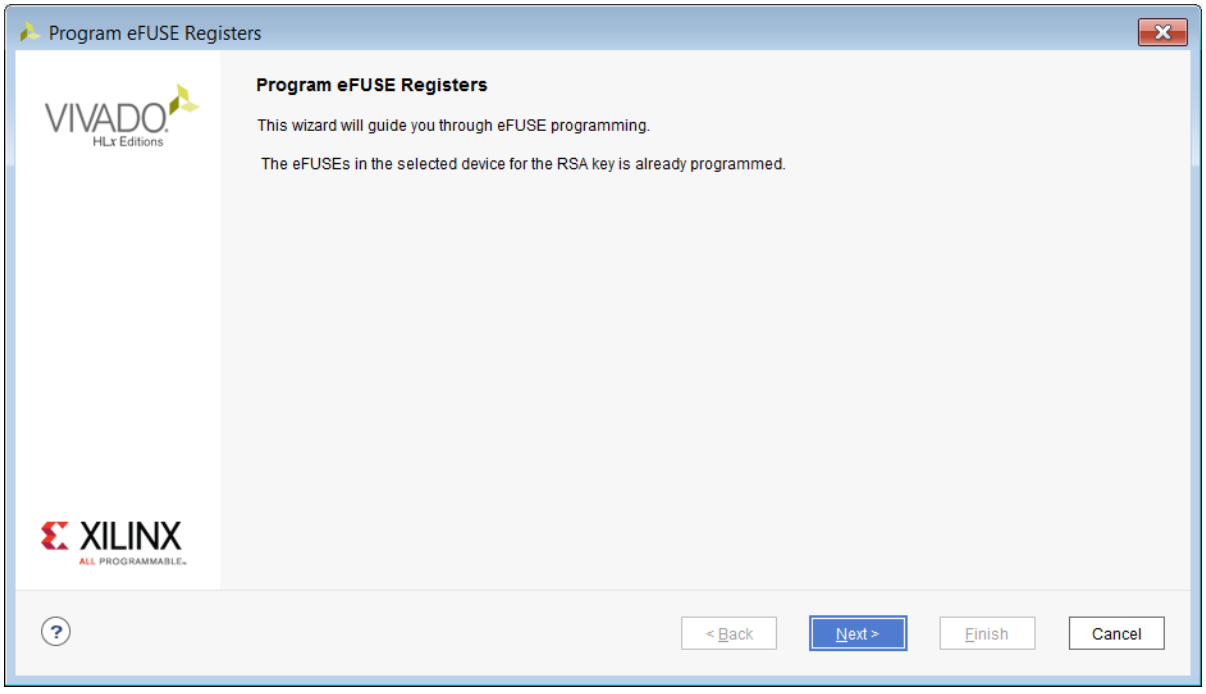
要执行 eFUSE 寄存器烧录，请在“Hardware”（硬件）窗口中右键单击 FPGA 并选中“Program eFUSE Registers”（eFUSE 寄存器烧录）。

图 49：**选择“Program eFUSE Registers”（UltraScale 和 UltraScale+ 器件）**



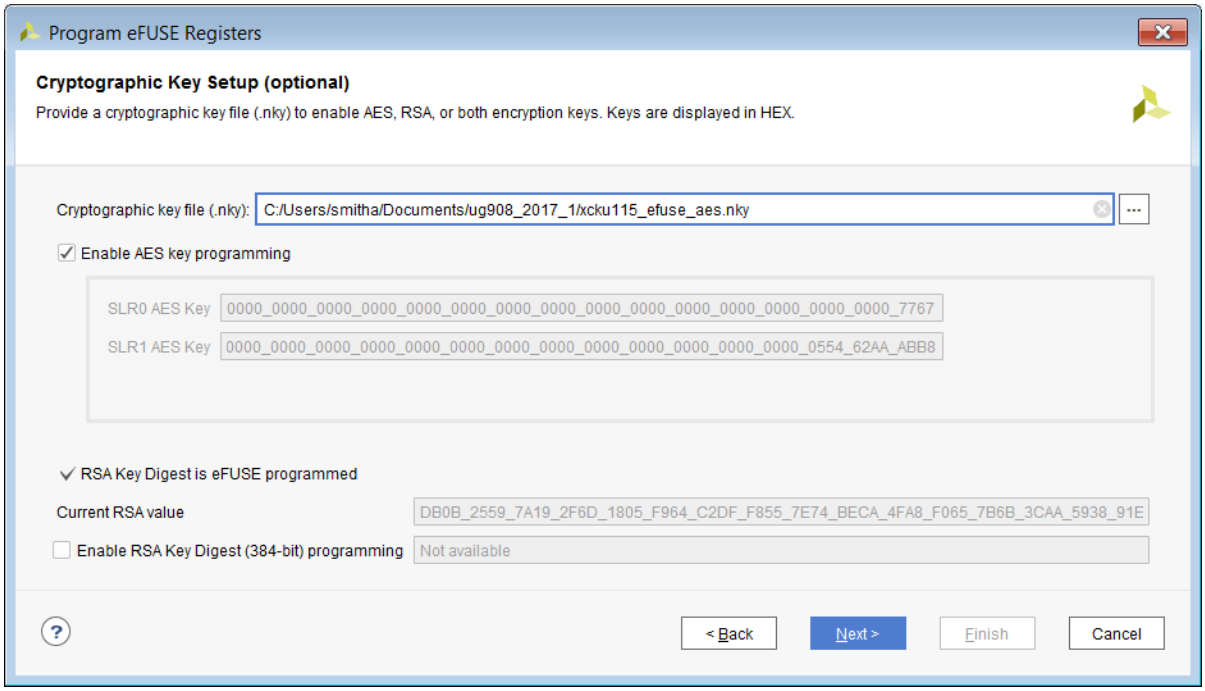
这样会显示如下图所示的“eFUSE Registers”Wizard（eFUSE 寄存器向导），此向导将指导您为 eFUSE 寄存器设置各选项。

图 50：**“Program eFUSE Registers”Wizard**



在“AES Key Setup”（AES 密钥设置）窗格中，指定以下设置：

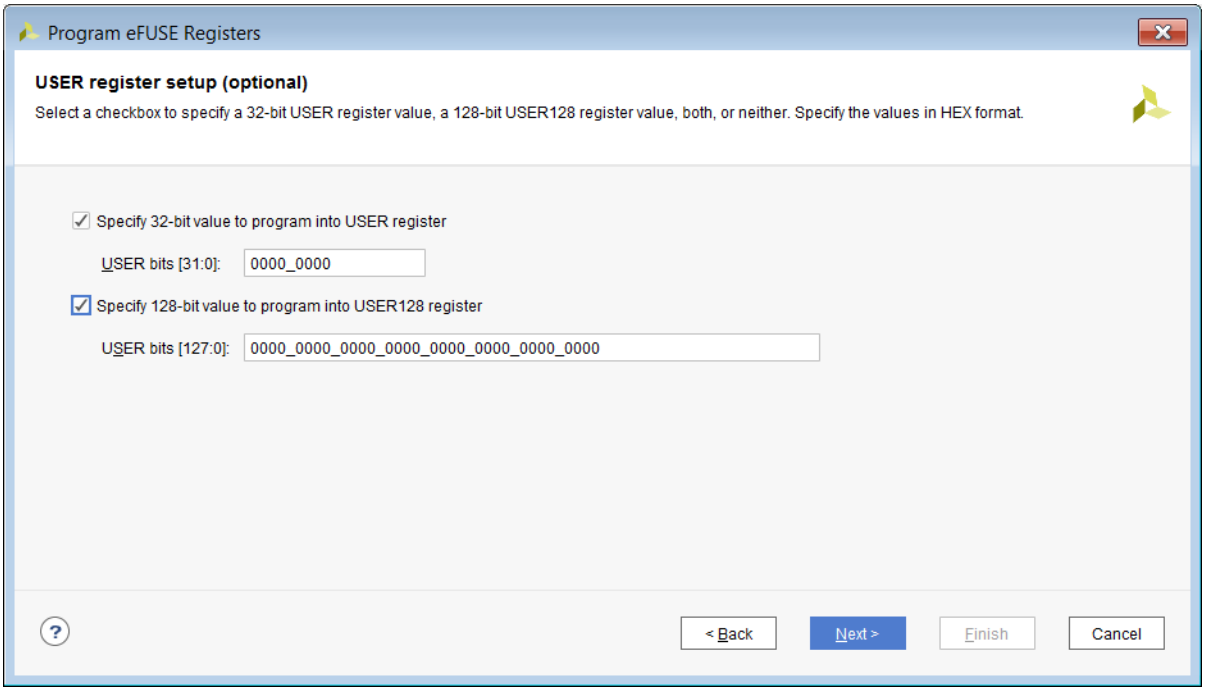
图 51：**eFUSE 加密密钥设置**



在“Cryptographic Key Setup”Wizard（加密密钥设置向导）窗格中，请指定如下密钥设置：

- 加密文件密钥 (NKY)：指定包含 eFUSE AES 和 RSA 密钥的 `.nky` 文件。
- AES 密钥（256 位）：256 位 AES eFUSE 密钥从指定 `.nky` 文件读入，用于对加载的加密比特流进行解密。
- RSA 密钥摘要（384 位）：384 位 RSA eFUSE 密钥从指定 `.nky` 文件读入，供 RSA 使用。
- 在“USER Register Setup”Wizard（用户寄存器设置向导）窗格中，指定 32 位用户寄存器或 128 位用户寄存器。

图 52：**eFUSE 用户寄存器设置**



在“USER Register Setup”窗格中，请指定用户定义的寄存器位数。32 位用户寄存器 (FUSE_USER) 和 128 位用户寄存器 (FUSE_USER128) 由一组用户定义的一次性可烧录 eFUSE 位组成。这些寄存器的位属于可累积烧录位。这表示如果您在任一 eFUSE 烧录会话中仅对 1 个 USER 位进行烧录（例如 USER = 0x0000_0001 或位 0），那么在后续 eFUSE 烧录会话中，您可对剩余 0 位的任意一位进行烧录（例如 USER = 0x0000_0002 或位 1）。

对 FUSE_USER 和 FUSE_USER_128 寄存器完成烧录后，可通过多种方式读入这些寄存器：

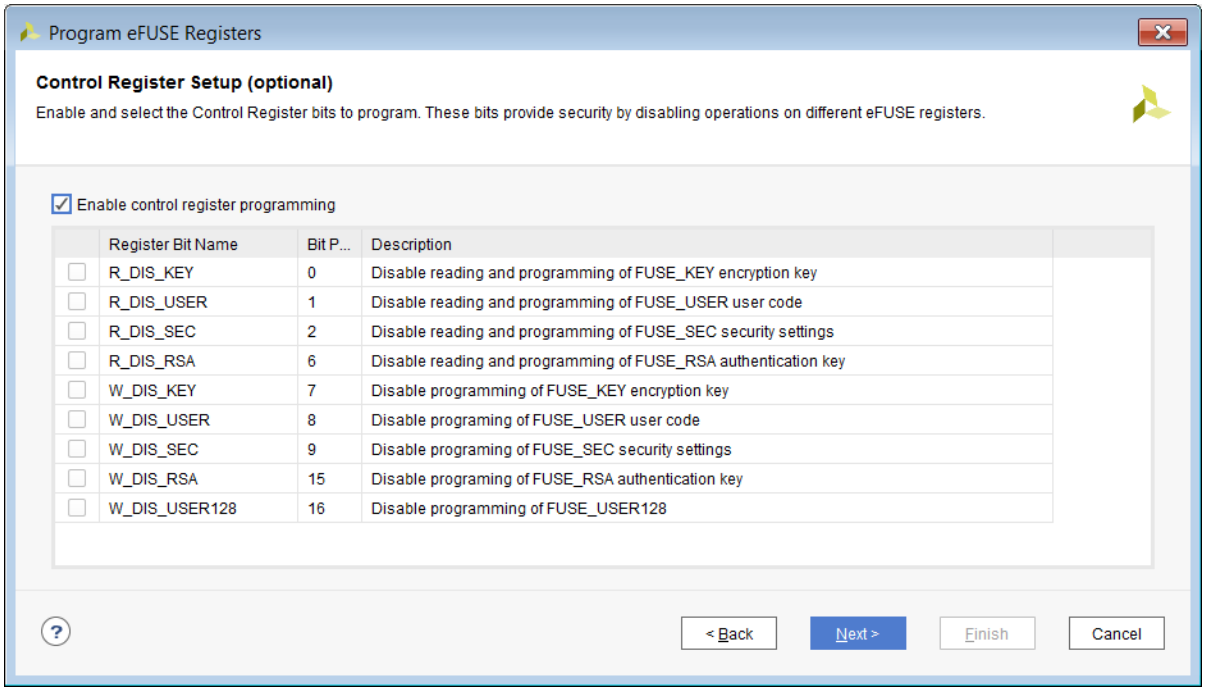
- 使用 Tcl 命令

```
report_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_USER
report_property [lindex [get_hw_devices] 0] REGISTER.EFUSE.FUSE_USER_128
```

- 运行 `refresh_hw_device` 操作后，通过 Vivado Hardware Device 的“Properties”（属性）窗口。

在“Control Register Setup”Wizard（控制寄存器设置向导）窗格中，指定以下设置：

图 53：“Control Register Setup”窗格



在“Control Register Setup”（控制寄存器设置）窗格中，指定 eFUSE 控制设置。

- R_DIS_KEY：设置此项即可禁用用于验证 FUSE_KEY 加密密钥的密钥和烧录操作的 CRC 检查。
- R_DIS_USER：设置此项即可禁用 32 位用户位 (FUSE_USER) 的读取和烧录操作。
- R_DIS_SEC：设置此项即可禁用安全寄存器位 (FUSE_SEC) 的读取和烧录操作。
- R_DIS_RSA：设置此项即可禁用 RSA 密钥寄存器 (FUSE_RSA) 的读取和烧录操作。
- W_DIS_USER：设置此项即可禁用 32 位用户位 (FUSE_USER) 的烧录操作。
- W_DIS_SEC：设置此项即可禁用安全寄存器位 (FUSE_SEC) 的烧录操作。
- W_DIS_RSA：设置此项即可禁用 RSA 密钥寄存器 (FUSE_RSA) 的烧录操作。
- W_DIS_USER_128：设置此项即可禁用 128 位用户位 (FUSE_USER128) 的烧录操作。

如需了解有关 FUSE_SEC 寄存器的更多详细信息，请参阅《UltraScale 架构配置用户指南》(UG570)。

禁用控制寄存器设置

要禁用控制寄存器烧录，请运行以下 Tcl 命令：

```
program_hw_devices -control_efuse {20} [lindex [get_hw_devices] $deviceIdx]
```

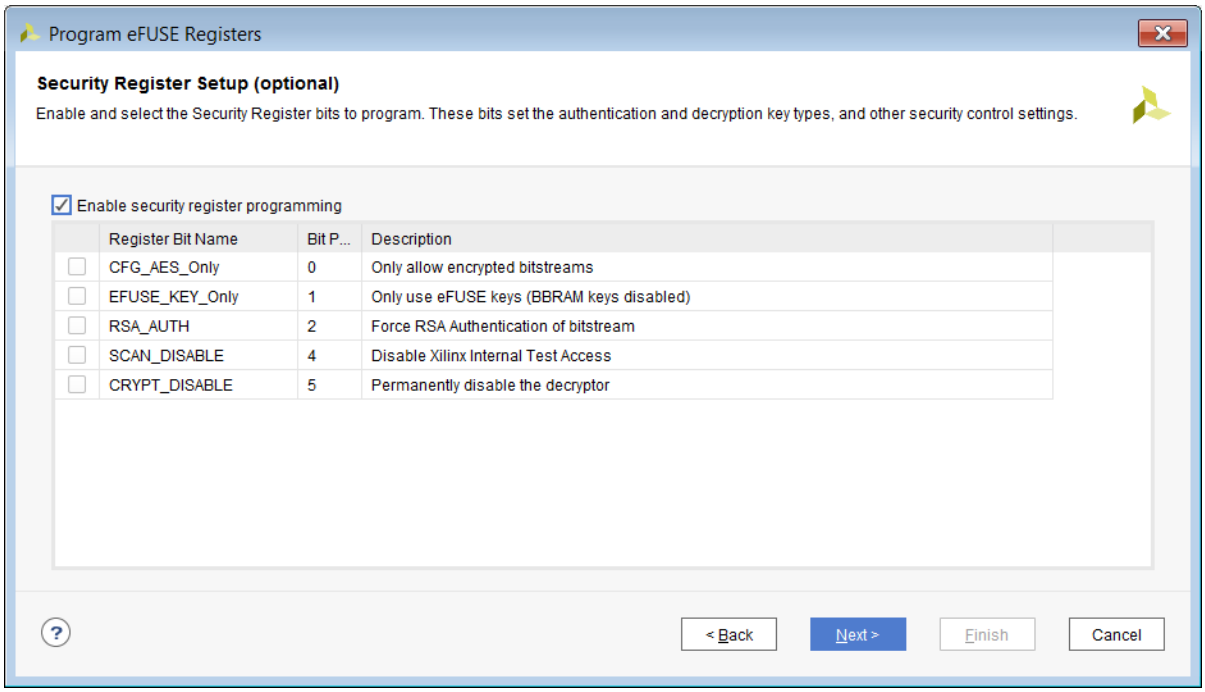
其中 `$deviceIdx` 设为 UltraScale 或 UltraScale+ 器件的索引，您可在此类器件中禁用 eFUSE 控制寄存器位烧录。

这样即可设置 W_DIS_CNTL 位，从而禁用后续 eFUSE 控制寄存器位烧录。

重要提示！如果 W_DIS_CNTL 位已烧录，那么将禁用其他 eFUSE 控制寄存器位的烧录操作，从而阻止对器件的控制寄存器进行进一步编辑。

在“Security Register Setup”Wizard（安全寄存器设置向导）窗格中，指定以下设置：

图 54：**eFUSE 安全寄存器设置**



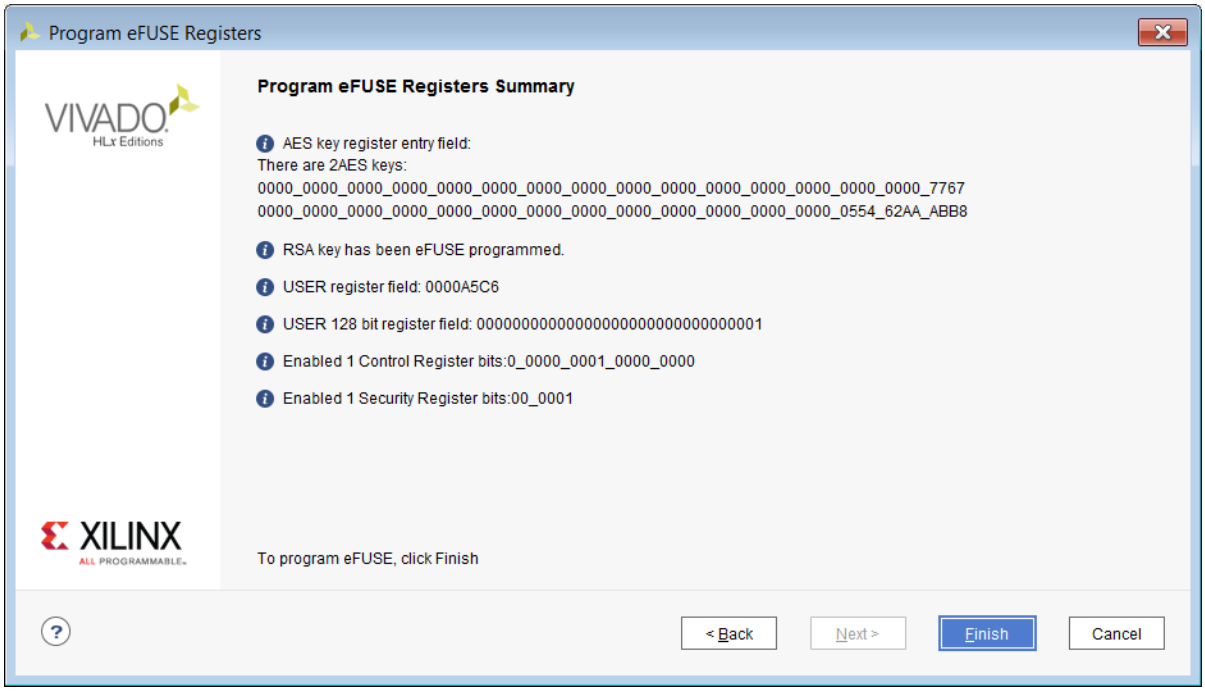
在“Security Register Setup”Wizard 窗格中，基于允许在 FPGA 上加载的比特流类型来指定安全控制选项。FUSE_SEC 设置如下：

- “CFG_AES_Only”：设置此项表示仅接受加密比特流。
- “EFUSE_KEY_Only”：设置此项表示仅限 eFUSE 密钥可用于解密。
- “RSA_AUTH”：设置此项表示强制对比特流执行 RSA 身份验证。
- “SCAN_DISABLE”：设置此项表示禁用 AMD 访问内部测试寄存器的权限。
- “CRYPT_DISBALE”：设置此项表示永久禁用解密器。

如需了解有关 FUSE_SEC 寄存器的更多详细信息，请参阅《UltraScale 架构配置用户指南》(UG570)。

请复查“Program eFUSE Registers Summary”（eFUSE 寄存器烧录汇总）窗格中的 eFUSE 设置。

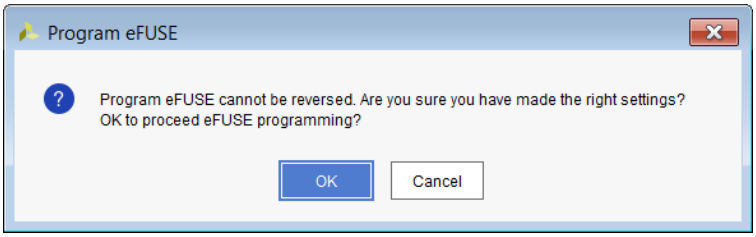
图 55：**Program eFUSE Registers Summary**



在此窗格中可显示“Program eFUSE Registers”Wizard（eFUSE 寄存器烧录向导）面板中设置的所有位。在此窗格中，您可查看各个位的设置，以便复查特定烧录设置。请仔细复查此汇总页面，以确保要烧录的每个位都已正确设置。

单击“Finish”（完成）即可显示“Program eFUSE”（eFUSE 烧录）确认对话框：

图 56：**“Program eFUSE”确认对话框**



单击“OK”（确定）以对指定的 FUSE 位进行烧录。

禁用 JTAG 接口

要通过 eFUSE 寄存器禁用 JTAG 接口，请运行以下 Tcl 命令：

```
program_hw_devices -force_efuse -security_efuse {08} [lindex
[get_hw_devices] $deviceIdx]
```

其中，`$deviceIdx` 设为已禁用 JTAG 接口的 UltraScale 或 UltraScale+ 器件的索引。

重要提示！用于为 7 系列器件禁用 JTAG 接口的 Tcl 命令不同于先前用于 UltraScale 或 UltraScale+ 器件的命令。如果使用 7 系列器件，请参阅答复记录 65110 中列出的 `XSK_EFUSEPL_DISABLE_JTAG_CHAIN` 条目。

注释：此烧录步骤应在所有期望的 eFUSE 位完成烧录后，作为最后一步来执行。

重要提示！如果 JTAG 禁用位已烧录，则将禁用 JTAG 接口，这会导致后续无法再接入器件进行测试和配置。仅当无需再通过 JTAG 访问器件时，才能对该位进行烧录。

强制执行 eFUSE 烧录

要对位于寄存器中任意位置的任何位（无论该位先前是否已烧录）强制执行置位，可将 `-force_efuse` 选项设置为 `program_hw_devices`。使用该选项时，将仅执行基本寄存器边界检查。

eFUSE NKZ 文件

为了将所有 eFUSE 烧录设置都捕获到一个文件中，以便于将 eFUSE 设置导出到其他 eFUSE 烧录器实现中，以及将这些 eFUSE 设置批量烧录到大量器件中，AMD 定义了称为 NKZ 的文件格式，并以文件扩展名 `.nkz` 来指定此格式。NKZ 格式是现有 NKY 格式的超集；即，NKZ 支持所有 NKY 字段以及任何可烧录 eFUSE 寄存器设置。

eFUSE 导出 NKZ 文件

由于建议您分多阶段烧录 eFUSE 设置，因此 eFUSE 设置始终导出至外部可见 NKZ 文件，并在每次 eFUSE 操作期间更新此文件。此文件用于累积应用于器件的所有 eFUSE 设置，即使通过多次 eFUSE 操作来应用这些设置也是如此。Vivado 会跟踪此文件，确保始终有单一 eFUSE 导出文件，其中包含 NKZ 格式的器件累积 eFUSE 设置。

如不指定任何选项，则此文件的默认名称如下所示，其中 `<FUSE_DNA>` 是器件的 FUSE_DNA 寄存器值。

```
export_<FUSE_DNA>.nkz
```

此默认文件位于 Vivado IDE 的启动目录中。可使用 `program_hw_devices` Tcl 命令搭配 `-efuse_export_file` 选项来更改此文件，如以下示例所示：

```
program_hw_devices -user_efuse {1} -efuse_export_file {my_settings.nkz}
```

Vivado IDE 会使用针对导出的 eFUSE 设置指定的 NKZ 文件来启动，而无需在 Tcl 选项中反复指定此文件。这样就不会移除先前创建的任何 eFUSE 导出文件，而是在此文件中包含更改 eFUSE 导出文件名之前应用的所有 eFUSE 设置。执行完所有 eFUSE 操作后，所有 eFUSE 设置都存储在当前 eFUSE 导出文件中。

此外还有 1 个选项可用于单独导出 eFUSE 设置，而无需实际完成器件烧录操作。该选项可用于创建包含所有必需 eFUSE 设置的单一 NKZ 文件，而不会影响器件。通过直接移除 eFUSE 导出文件并从头开始操作即可轻松解决在此模式下发生的任意 eFUSE 烧录错误。可通过以下 Tcl 命令来使用此仅限导出模式：

```
program_hw_devices -user_efuse {1} -only_export_efuse
```

该选项仅适用于单一命令，因此必须在每项 eFUSE 操作中使用该选项，以避免烧录器件。由于 eFUSE 设置始终导出至 NKZ 文件，与是否执行烧录无关，因此建议不要将烧录流程与仅限导出流程混用。否则，生成的 eFUSE 导出文件可能同时包含实际已烧录到器件中的 eFUSE 设置和仅导出到 NKZ 文件的设置，导致器件中 eFUSE 寄存器状态不明确。通过使用此仅限导出模式，所创建 NKZ 格式的 eFUSE 导出文件与执行烧录后创建的导出文件完全相同，而无需在器件上执行烧录。

此外，在 `program-hw_devices` Tcl 命令上还可使用 `-skip_program_keys` 选项来跳过 NKZ 文件中指定的加密密钥烧录操作。

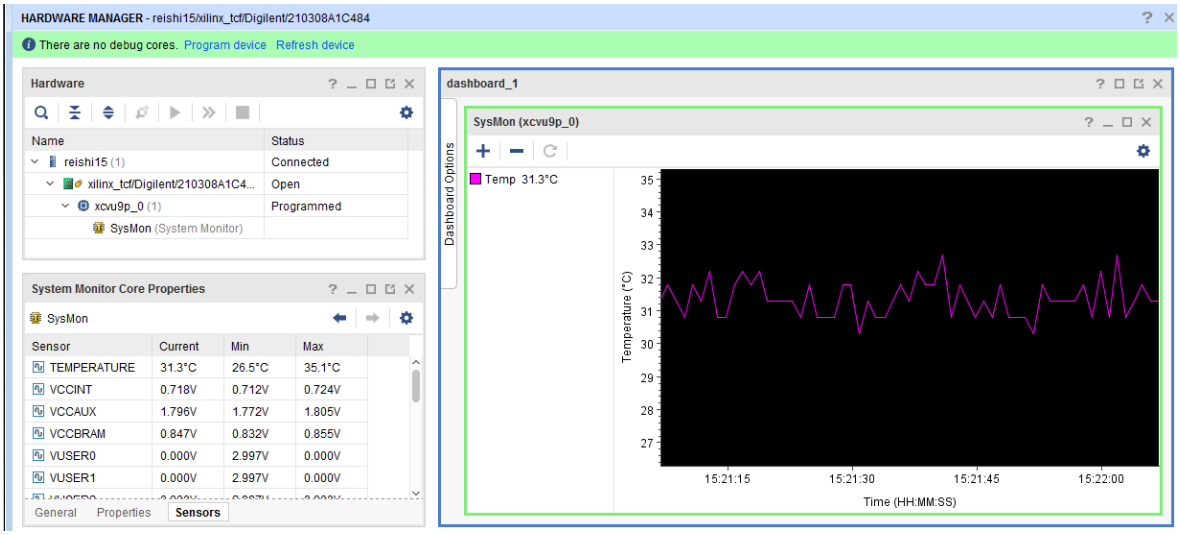
- 对于 7 系列器件，如果指定 `-skip_program_keys`，则跳过 `FUSE_AES` 的烧录，不执行任何检查来确认加密文件中的 AES 密钥，并持续进行烧录。
- 对于 7 系列器件，如果 NKZ 同时包含非零 `FUSE_USER` 设置，则不得使用 `-skip_program_keys`，因为 `FUSE_AES` 值与 `FUSE_USER` 值必须在 7 系列器件中一起进行烧录。
- 对于 UltraScale 器件和 UltraScale+ 器件，如果进行模糊密钥烧录时使用 `-skip_program_keys`，则跳过 `FUSE_SEC[6]` 的设置。

系统监控器

系统监控器 (SYSMON) 模数转换器 (ADC) 用于测量硬件器件上的裸片温度和电压。SYSMON 可通过片上温度和供电传感器来监控物理环境。ADC 可为各种应用提供高精度模拟接口。请参阅下文，以获取有关特定器件架构的更多信息。

- 《UltraScale 架构系统监控器用户指南》(UG580)
- 《7 系列 FPGA 与 Zynq 7000 SoC XADC 双 12 位 1 MSPS 模数转换器用户指南》(UG480)
- 《Versal 自适应 SoC 系统监控器架构手册》(AM006)

图 57：**系统监控器**



hw_sysmon 数据存储在称为状态寄存器的专用寄存器中，此类寄存器可通过 hw_sysmon_reg 对象来访问。您可使用 `get_hw_sysmon_reg` 命令来获取系统监控器寄存器的内容。

调用 `refresh_hw_device` 时，支持系统监控器的每个器件都会自动创建 1 个或多个 `hw_sysmon` 对象。创建 `hw_sysmon` 对象时，会为其分配 1 个属性，该属性适用于所有温度和电压寄存器以及控制寄存器。在 `hw_sysmon` 对象上，分配给温度和电压寄存器的值已转换为摄氏度/华氏度和伏特值。

虽然您可使用 `get_hw_sysmon_reg` 命令来访问系统监控器的寄存器中存储的十六进制值，但您也可通过 `hw_sysmon` 对象的格式化属性来检索某些寄存器的值。例如，以下代码用于检索指定 `hw_sysmon` 对象的 TEMPERATURE 属性，而不是直接访问寄存器的十六进制值：

```
set opTemp [get_property TEMPERATURE [lindex [get_hw_sysmons] 0]
```

在 hw_sysmon Tcl 命令描述 中可找到所有系统监控器命令的完整列表。

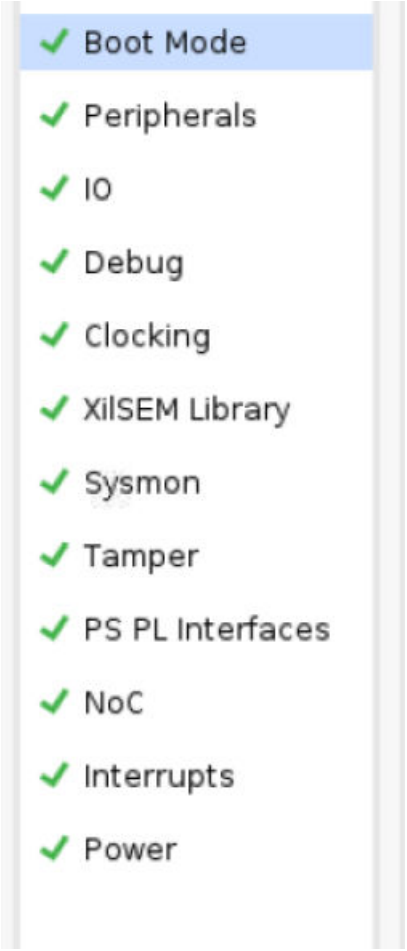
适用于 Versal 器件的系统监控器

不同于先前架构，AMD Versal™ 器件上使用的系统监控器可以显示大量片上传感器。在 Versal 器件上使用系统监控器之前，必须在“PS-PMC”设置下选择要在 Control, Interfaces, and Processing System (CIPS) IP 中测量的传感器。

注释：如果未选择任何传感器或者如果未配置 CIPS，则只能获取器件温度。

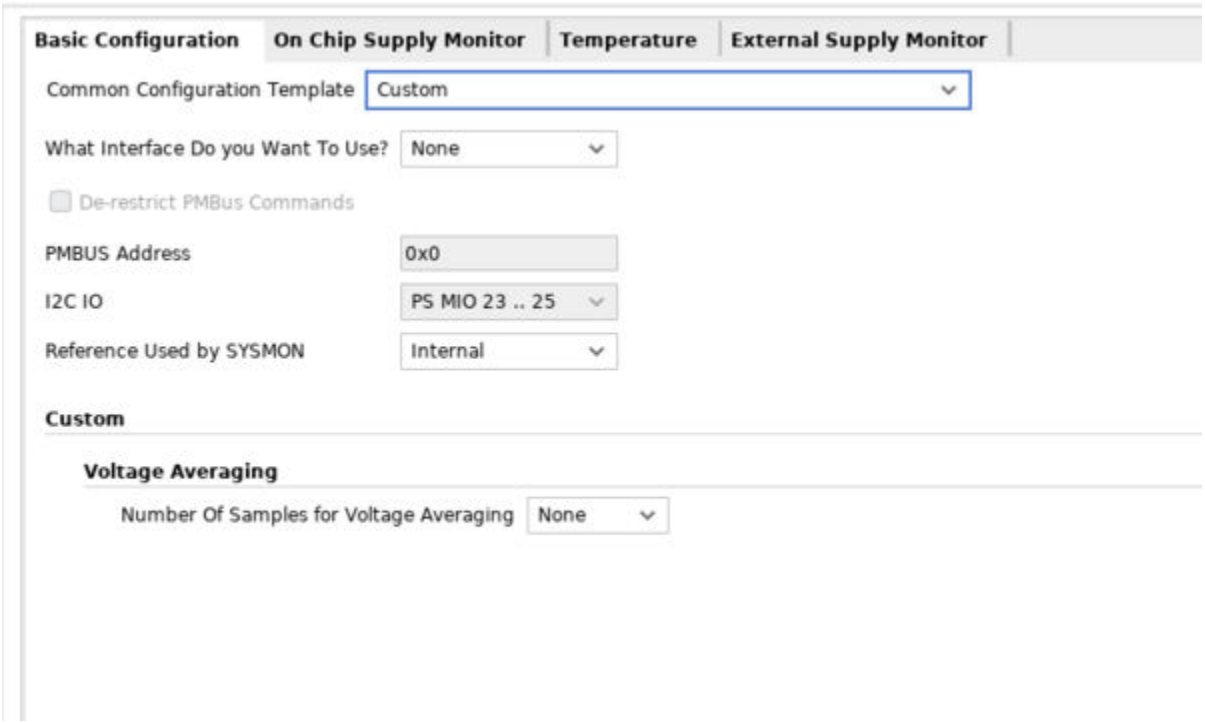
在 Versal CIPS IP 中配置系统监控器传感器

1. 确保在设计中已例化 CIPS。如需了解有关集成 CIPS IP 的更多信息，请参阅《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)。如果设计中已存在 CIPS，请单击 Flow Navigator 的 IP integrator 下的“Open Block Design”（打开块设计）以打开包含此 CIPS 的块设计。打开块设计后，双击“CIPS”以启动 IP 自定义 GUI。
2. 单击“Next”（下一步），然后单击带有“PS PMC”标记的蓝色框以进入“PS-PMC configuration”Wizard（PS-PMC 配置向导）。
3. 在左侧窗格中，单击“Sysmon”。



4. 这样会显示“SysMon Configuration”（系统监控器配置），此时即可对系统监控器进行配置。选中“Common Configuration Template”（常用配置模板）即可快速配置基本温度和/或电压监控。要选择特定电压轨以便进行监控，请单击“On Chip Supply Monitor”（片上电源监控）选项卡，然后勾选要监控的电压轨旁的“Enable”（启用）单选按钮。

图 58：**在 CIPS IP 自定义 GUI 中配置系统监控器传感器**



第 9 章

标准测试和编程语言 (STAPL) 编程

注释：在 AMD UltraScale™ /AMD UltraScale+™ FPGA 和 AMD Versal™ 器件上支持 STAPL 编程。在 AMD Zynq™ 器件和 7 系列 FPGA 上不支持 STAPL 编程。

标准测试和编程语言 (STAPL) 文件也可用于对 FPGA 和配置存储器器件进行烧录。通过 AMD Vivado™ Design Suite 和 AMD Vivado™ Lab Edition 生成的 STAPL 文件包含对这些器件执行烧录所需的低级别 JTAG 指令和数据。生成此文件后，边界扫描测试工具即可脱离 Vivado IDE 来单独使用。

创建 STAPL 文件的一般步骤如下所述：

1. 创建 STAPL 目标。
2. 向 STAPL 目标添加器件。
3. 为 STAPL 链中的器件添加操作。
4. 写入 STAPL 文件。
5. 关闭 STAPL 目标。
6. （可选）执行 STAPL。

步骤 4 按顺序记录烧录操作，并将其存储为缓存文件。此缓存文件将写出至步骤 5 中的目标。此文件创建完成后，即可通过边界扫描工具来使用，或者通过 Vivado Design Suite 或 Vivado Lab Edition 工具来执行。

创建 STAPL 目标

STAPL 目标类似于有效的 AMD 平台电缆 USB 或 Digilent JTAG 线硬件目标。属性和 Tcl 命令全部相同，主要差异在于 STAPL 目标并非实时有效的电缆。对此目标执行的任何操作直至执行 STAPL 后才会对硬件产生影响。

注释：您无需将电缆连接至系统以创建 STAPL。

使用命令行

以下是初始启动 Vivado 或 Vivado Lab Edition 后创建 STAPL 目标所需的步骤：

```
open_hw_manager connect_hw_server create_hw_target -stapl my_stapl_target
if {[string length [get_hw_targets -quiet -filter {IS_OPENED == TRUE}]] >
0} \ {close_hw_target [get_hw_targets * -filter {IS_OPENED == TRUE} ] }; \
open_hw_target [get_hw_targets -regexp .*my_stapl_target] current_hw_target
```

如已连接到服务器，则可省略前 2 条命令。执行 `create_hw_target` 命令可定义 `my_stapl_target`。

注释：同一会话中不得存在 2 个同名目标。

最后，关闭所有打开的目标并打开 STAPL 目标后，就会运行 create_hw_target 命令。这样最终命令会显示已创建的 my_stapl_target 的完整硬件目标处理名称。

针对目标执行的所有标准操作（例如，get_hw_targets 和 open_hw_target 命令）均受支持。您可使用 IS_STAPL 硬件目标属性来区分活动目标和 STAPL 目标。例如，以下命令行样本可从名为“my_stapl_target”的目标读取 IS_STAPL 属性。

```
get_property IS_STAPL [get_hw_targets -regexp .*my_stapl_target]
```

此外，发出以下命令即可显示此会话中创建的所有 STAPL hw_targets：

```
get_hw_targets -filter {IS_STAPL}
```

要删除已创建的目标，请使用 delete_hw_target 命令。例如，发出以下命令即可删除 my_stapl_target：

```
delete_hw_target [get_hw_targets -regexp .*my_stapl_target]
```

重要提示！删除目标后，还会删除针对此目标创建的所有器件。此外，如果删除的目标先前处于打开状态，则会被关闭。

向 STAPL 目标添加器件

创建 STAPL 目标后，可向其中添加器件以定义 STAPL JTAG 器件链配置。STAPL JTAG 器件链配置应与目标硬件链相匹配，以确保能正确执行 STAPL 文件。

注释：不建议创建同时包含 UltraScale/UltraScale+ FPGA 和 Versal 器件的 STAPL 器件链，因为这可能导致无法预测的结果。在后续版本中将着力解决此限制。

使用命令行

要在 Vivado IDE 中使用 Vivado Tcl 模式或 Tcl 控制台来创建 JTAG 链，请在已打开的 STAPL 目标上按顺序执行 create_hw_device 操作。例如，要先添加 xcvc1902 部件，然后添加 xcvu095 部件，请执行以下步骤：

```
current_hw_target my_stapl_target open_hw_target create_hw_device -part
xcvc1902 create_hw_device -part xcvu095 refresh_hw_target get_hw_devices
```

在此示例中，如果已创建并打开 STAPL，则可跳过前 2 个步骤。此示例中的 create_hw_device 命令用于定义 JTAG 链中的器件，从该链上的首个器件开始。

注释：create_hw_device 命令仅在已打开的 STAPL 硬件目标上创建器件。

注释：器件只能添加到 STAPL 器件链中。

要将用户定义的器件添加到此链中，请使用 -part 选项添加 -idcode、-irlength 和 -mask 值以及部件类型名称。例如，如果部件名为“my_part”，JTAG idcode 为 1234567、“ir length”为 8 且 mask 为 ffffffff，请按如下所示创建器件：

```
open_hw_target [current_hw_target] create_hw_device -idcode 01234567 -
irlength 8 -mask ffffffff -part my_part # print IR length for user defined
devices puts [get_property IR_LENGTH [lindex [get_hw_devices -filter {PART
== my_part}] 0]] puts $idcode_hex close_hw_target
```

注释： `create_hw_device` 的 `idcode` 应为有效的器件 ID 代码。硅片供应商通常通过器件 BSDL 文件来提供 ID 代码值和 IR 长度。

要查看目标及其器件的报告，请运行 `report_hw_targets` 命令。此报告可显示系统中所有活动目标的详细信息。此报告可用于获取服务器、目标和器件的属性，如下所示：

```
report_hw_targets INFO: Server Property Information: localhost:3121 CLASS:
hw_server HOST: localhost NAME: localhost:3121 PORT: 3121 SID:
TCP:localhost:3121 INFO: Target Property Information: localhost:3121/
xilinx_tcf/Xilinx/my_stapl_target CLASS: hw_target DEVICE_COUNT: 3 HW_JTAG:
0 IS_OPENED: 1 MAX_DEVICE_COUNT: 32 NAME: localhost:3121/xilinx_tcf/Xilinx/
my_stapl_target FREQUENCY: 10000000 TYPE: xilinx_tcf TID: jsn-XNC-
my_stapl_target UID: Xilinx/my_stapl_target SVF: 0 STAPL: 1 Device:
arm_dap_0 Device: xcvc1902_1 Device: xcvu095_2 INFO: Device Property
Information: arm_dap_0 CLASS: hw_device PART: arm_dap ID CODE:
01101011101000000000010001110111 IR LENGTH: 4 MASK: 0 USER_CHAIN_COUNT: 0
PROGRAMMING FILE: PROBES FILE: PROGRAMMING STATUS: 0 INFO: Device Property
Information: xcvc1902_1 CLASS: hw_device PART: xcvc1902 ID CODE:
00010100110010101000000010010011 IR LENGTH: 6 MASK: 0 USER_CHAIN_COUNT: 4
PROGRAMMING FILE: PROBES FILE: PROGRAMMING STATUS: 0 INFO: Device Property
Information: xcvu095_2 CLASS: hw_device PART: xcvu095 ID CODE:
01110011100001000010000010010011 IR LENGTH: 6 MASK: 0 USER_CHAIN_COUNT: 4
PROGRAMMING FILE: PROBES FILE: PROGRAMMING STATUS: 0
```

注释： 在 STAPL 链中不支持向器件添加配置存储器部件。

有关 STAPL 链的操作

按正确顺序创建反映所有器件的 STAPL 链之后，即可开始向 STAPL 链中的器件添加烧录操作。

使用命令行

以下命令可用于对 STAPL 链中的器件进行烧录：

```
create_hw_target -stapl my_stapl_target open_hw_target set device0
[create_hw_device -part xcvc1902] set_property PROGRAM.FILE
{my_xcvc1902.pdi} $device0 program_hw_devices $device0
```

写入 STAPL 文件

使用命令行

要在 Vivado IDE 中使用 Vivado Tcl 模式或者 Tcl 控制台来编写 STAPL 文件，请使用 `write_hw_stapl` 命令。

这样会在临时文件中捕获 STAPL 链的 FPGA/SoC 直接操作。调用 `write_hw_stapl` 命令时，临时文件将改为传递给该命令的文件名。调用 `write_hw_stapl` 命令后，临时文件将复位，并在 STAPL 文件序列开头处添加 1 项后续烧录操作。

以下代码段显示了用于创建名为 `my_vc1902.stapl` 的文件的 Tcl 命令（包括对 xcvc1902 器件进行直接烧录）：

```
create_hw_target -stapl my_stapl_target open_hw_target set device0
[create_hw_device -part xcvc1902] set_property PROGRAM.FILE
{my_xcvc1902.pdi} $device0 program_hw_devices $device0 current_hw_device
[lindex [get_hw_devices] 1] write_hw_stapl my_xcvc1902.stapl close_hw_target
```

在此样本代码中，xcvc1902 器件是使用 `create_hw_device` 命令创建的，其返回值设为名为 device0 的临时变量。将 PROGRAM.FILE 属性设为 `my_xcvc1902.pdi` 文件时，此临时值将用于引用对象。下一步，将使用 device0 引用来调用 `program_hw_device` 命令。运行此 `program_hw_device` 命令时，它会通过必要的 STAPL 操作来创建临时 STAPL 文件，用于对 xcvc1902 上的 `my_xcvc1902.pdi` 文件执行烧录。最后，`write_hw_stapl` 命令会将此临时文件移至最终目标 `my_xcvc1902.stapl`。此时，STAPL 文件创建进程即告完成，并且可关闭目标。

提示：编写 STAPL 文件时，应首先为 JTAG 链创建所有器件，然后再执行烧录操作。如果在执行烧录命令间交织执行了 `create_hw_device` 命令，那么生成的输出 STAPL 文件会包含 2 条不同的序列链。

错误的 STAPL 文件创建步骤示例：

```
create_hw_target -stapl my_stapl_target open_hw_target set device0
[create_hw_device -part xcvc1902] set_property PROGRAM.FILE
{my_xcvc1902_1.pdi} $device0 # this program command will produce stapl
instructions # which account for only device0 in chain program_hw_devices
$device0 set device1 [create_hw_device -part xcvc1902] set_property
PROGRAM.FILE {my_xcvc1902_2.pdi} $device1 # this program command will
produce stapl instructions # which account for device0 and device1 in chain
program_hw_devices $device1 write_hw_stapl my_bad_xcvc1902.stapl
close_hw_target
```

第一条烧录命令仅捕获包含首个器件的链定义。第二条烧录命令在写出 STAPL 指令时会包含链中的 2 个器件。因此如果您尝试在含 2 个器件的链上运行此 STAPL 文件，则第一项烧录操作将失败，因为活动链会收到 2 个器件，而非此命令期望的 1 个器件。

要纠正此问题，请首先运行 `create_hw_device` 命令。当链完成定义后，请按如下所示执行烧录操作：

```
create_hw_target -stapl my_stapl_target open_hw_target # create device
chain first set device0 [create_hw_device -part xcvc1902] set device1
[create_hw_device -part xcvc1902] # program device0 set_property
PROGRAM.FILE {my_xcvc1902_1.bit} $device0 program_hw_devices $device0 #
program device1 set_property PROGRAM.FILE {my_xcvc1902_2.bit} $device1
program_hw_devices $device1 write_hw_stapl my_good_xcvc1902.stapl
close_hw_target
```

执行 STAPL 文件

创建 STAPL 文件后，您可选择通过 `stapl_player` 应用可执行文件来执行 STAPL 文件，在 Vivado 安装中包含此可执行文件。STAPL 播放器可以执行通过 STAPL 生成功能所生成的 STAPL 文件，主要用作为确认测试工具。`stapl_player` 应用并非通用的 STAPL 执行命令，只能使用通过 Vivado IDE 创建的 STAPL 文件。

要运行 STAPL 文件，您可以在活动目标上按如下方式使用命令来运行该应用：

```
stapl_player run --help Usage: stapl_player run [OPTIONS] Options: -i, --
input-file FILE STAPL file to play [required] -h, --hw-server TEXT Hardware
server URL [default: TCP:localhost:3121] -a, --action TEXT STAPL action to
perform [required] --cable-serial TEXT Serial number of JTAG cable. This
option can be used to choose a specific JTAG cable amongst multiple cables
connected to the hardware server. [optional] --help Show this message and
exit.
```

示例

```
$stapl_player run --input-file my_vc1902.stapl --hw-server
TCP:192.168.0.100:3121 --action PROGRAM ******** AMD stapl-player v2024.1.0
****** Build date : Mar 21 2024-07:45:59 **** Build number :
2024.1.1710987359 ** Copyright 2021-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2024 Advanced Micro Devices, Inc. All Rights Reserved.
```

在此示例中，在主机 192.168.0.100 上运行的硬件服务器上执行来自 `my_vc1902.stapl` 文件的 PROGRAM 操作。

如果运行该硬件服务器的主机具有多条 JTAG 线缆，那么您可使用 STAPL 播放器的 `list_cables` 命令来列出可用线缆的序号。您可使用 `run` 命令的 `--cable-serial` 选项来指定目标序号。

示例

```
$ stapl_player list_cables -h 192.168.0.200:3121 ******** Xilinx stapl-
player v2024.1.0 ****** Build date : Mar 21 2024-07:45:59 **** Build
number : 2024.1.1710987359 ** Copyright 2021-2022 Xilinx, Inc. All Rights
Reserved. ** Copyright 2022-2024 Advanced Micro Devices, Inc. All Rights
Reserved. Connecting to the hardware server at 192.168.0.200:3121 Serial
numbers of JTAG cables connected to the hardware server on
192.168.0.200:3121 are - Cable #1 - 492218147875A Cable #2 - 0000180fe3cf01
```

在此示例中，STAPL 播放器列出了 192.168.0.200 上可用的 2 条线缆的序号。

第 10 章

串行矢量格式 (SVF) 文件烧录

注释：串行矢量格式 (SVF) 烧录在 AMD Versal™ 器件上不受支持。

对 FPGA 和配置存储器器件进行烧录的另一种方法是通过使用串行矢量格式 (SVF) 文件来执行烧录。通过 AMD Vivado™ Design Suite 和 Vivado Lab Edition 生成的 SVF 文件包含烧录这些器件所需的低级别 JTAG 指令和数据。生成此文件后，即可通过独立于 Vivado IDE 的边界扫描测试工具来使用。

创建 SVF 文件的一般步骤如下所述：

1. 创建 SVF 脱机目标。
2. 打开创建的 SVF 目标。
3. 向目标添加器件以定义 SVF JTAG 扫描链。
4. 对 FPGA 或配置存储器器件进行烧录。
5. 编写 SVF。
6. 关闭 SVF 目标。
7. （可选）执行 SVF。

在步骤 4 中，烧录操作将按顺序被记录并存储为缓存文件。此缓存文件将写出至步骤 5 中的目标。此文件创建完成后，即可通过边界扫描工具来使用，或者通过 Vivado Design Suite 或 Vivado Lab Edition 工具来执行。

重要提示！XSVF 文件格式在 Vivado IDE 中不受支持。

创建 SVF 目标

SVF 目标类似于有效的 AMD 平台电缆 USB 或 Digilent JTAG 线硬件目标。属性和 Tcl 命令全部相同，主要差异在于 SVF 目标并非实时有效的电缆。对此目标执行的任何操作直至执行 SVF 后才会对硬件产生影响。

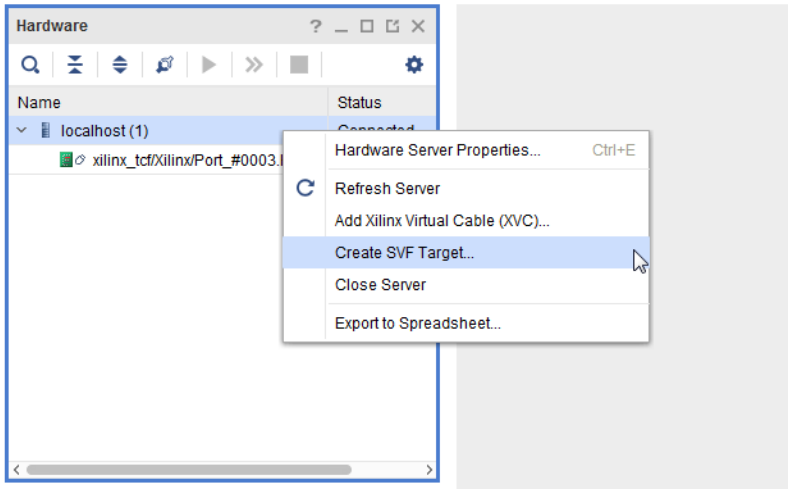
注释：您无需将电缆连接至系统以创建 SVF。

使用 Vivado IDE

要在 Vivado 硬件管理器中创建 SVF 目标，请通过启动 Vivado 或 Vivado Lab Edition 来打开 Vivado 硬件管理器。您可依次选择“Tools” → “Create SVF Target” （工具 > 创建 SVF 目标）来创建 SVF 目标。这样会在本地主机上自动打开服务器，还会打开“Create SVF Target”对话框，如下图中的对话框所示。

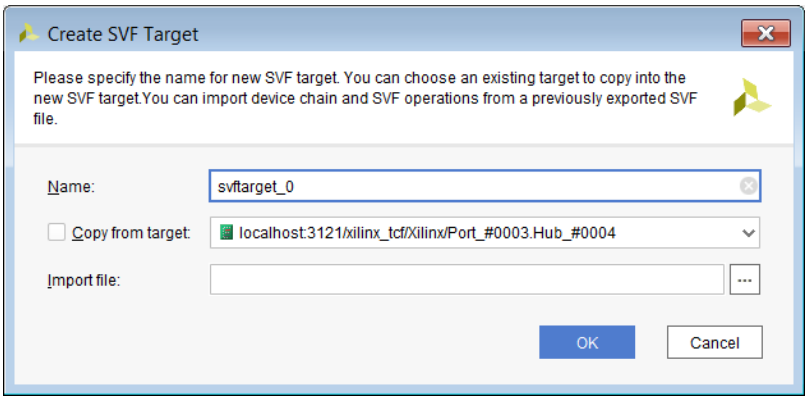
在任意可用服务器上，您均可创建脱机 SVF 目标，如下所示：

图 59：**Create SVF Target**



这样会打开“Create SVF Target”（创建 SVF 目标）对话框，如下所示：

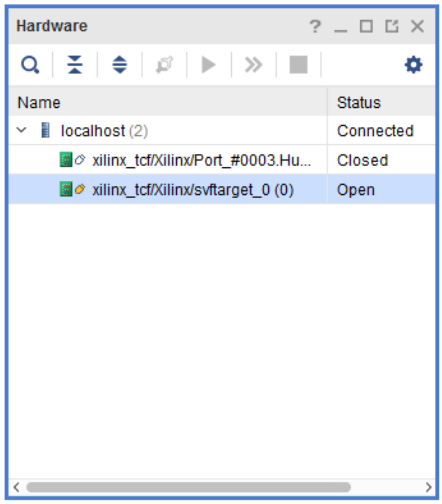
图 60：**“Create SVF Target”对话框**



提示：您可通过启用“Copy from target”（从目标复制）选项来复制现有 SVF 链。或者，您也可以指定在先前运行的流程中使用 Vivado 硬件管理器创建的 SVF 文件。Vivado IDE 会保存 SVF 链的规格，以便在回读时可重新创建该 SVF 链。

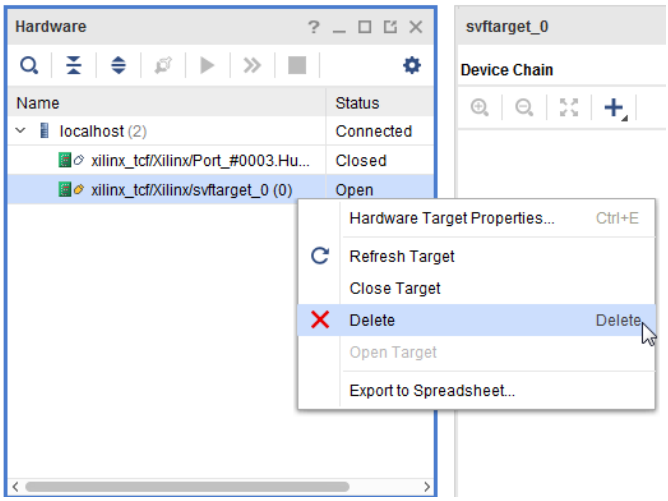
在 Vivado 硬件管理器的“Hardware”（硬件）窗口中您的服务器下，您创建的 SVF 目标会显示“Open”（打开）。

图 61：**“Hardware”窗口中的 SVF 目标**



要删除现有 SVF 目标，请在“Hardware”窗口中右键单击该 SVF 目标，然后选择“Delete”（删除）。

图 62：**在“Hardware”窗口中删除 SVF 目标**



重要提示！删除目标后，还会删除针对此目标创建的所有器件。此外，如果删除的目标先前处于打开状态，则会被关闭。

您也可以在 Vivado IDE 中使用 Vivado Tcl 模式或 Tcl 控制台来创建 SVF 目标。

以下是初始启动 Vivado 或 Vivado Lab Edition 后创建 SVF 目标所需的步骤：

使用命令行

以下是初始启动 Vivado 或 Vivado Lab Edition 后创建 SVF 目标所需的步骤：

```
open_hw_manager connect_hw_server create_hw_target my_svf_target if
{[string length [get_hw_targets -quiet -filter {IS_OPENED == TRUE}]] > 0} \
{close_hw_target [get_hw_targets * -filter {IS_OPENED == TRUE} ] }; \
open_hw_target [get_hw_targets *my_svf_target] current_hw_target
```

如已连接到服务器，则可省略前 2 条命令。执行 `create_hw_target` 命令可定义 `my_svf_target`。

注释：同一会话中不得存在 2 个同名目标。

最后，关闭所有打开的目标并打开 SVF 目标后，就会运行 create_hw_target 命令。这样最终命令会显示已创建的 `my_svf_target` 的完整硬件目标处理名称。

针对目标执行的所有标准操作（例如，`get_hw_targets` 和 `open_hw_target` 命令）均受支持。您可使用 `IS_SVF` 硬件目标属性来区分活动目标和 SVF 目标。例如，以下命令行样本可从名为“`my_svf_target`”的目标读取 `IS_SVF` 属性。

```
get_property IS_SVF [get_hw_targets -regexp .*my_svf_target]
```

此外，发出以下命令即可显示此会话中创建的所有 SVF `hw_targets`：

```
get_hw_targets -filter {IS_SVF}
```

要删除已创建的目标，请使用 `delete_hw_target` 命令。例如，发出以下命令即可删除 `my_svf_target`：

```
delete_hw_target [get_hw_targets -regexp .*my_svf_target]
```

重要提示！删除目标后，还会删除针对此目标创建的所有器件。此外，如果删除的目标先前处于打开状态，则会被关闭。

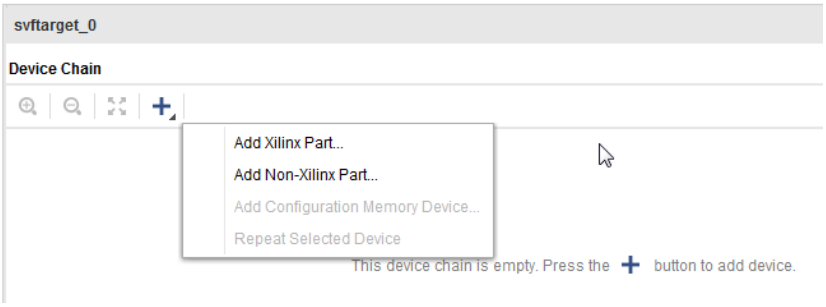
向 SVF 目标添加器件

创建 SVF 目标后，可向其中添加器件以定义 SVF JTAG 器件链配置。SVF JTAG 器件链配置应与目标硬件链相匹配，以确保能正确执行 SVF 文件。

使用 Vivado IDE

单击“+”按钮以向 SVF 链添加 AMD 部件或非 AMD 部件。

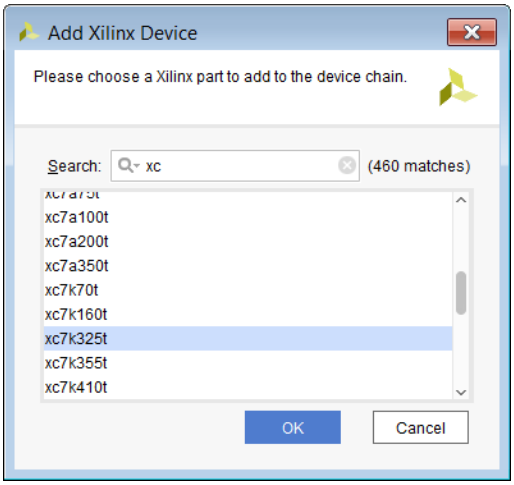
图 63：**向 SVF 目标添加器件**



单击“Add Xilinx Part”（添加 AMD 部件）时，会打开“Add Xilinx Device”（添加 AMD 器件）对话框。现在，您可选择相应的 AMD 器件以供添加到 SVF 链中。

注释：器件只能添加到 SVF 器件链中。

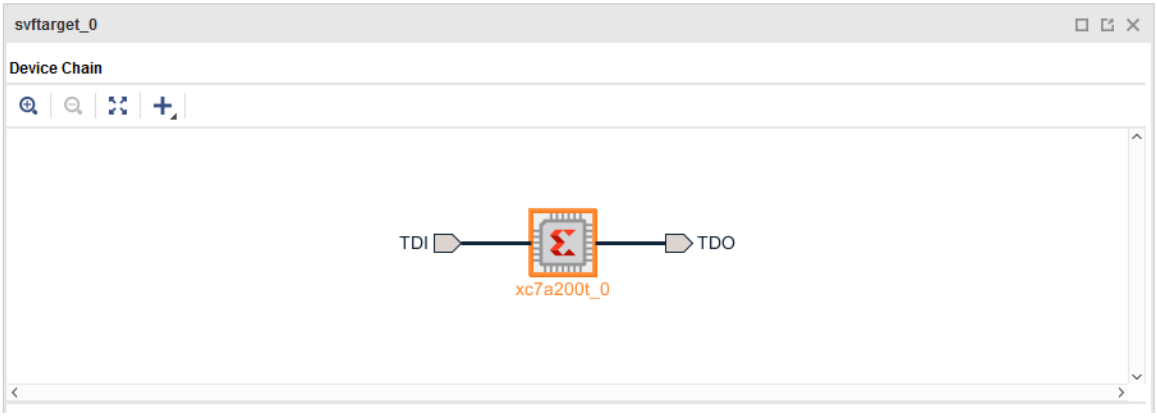
图 64：**“Add Xilinx Device”对话框**



提示：此对话框与 Vivado ML Enterprise 中所示略有不同。

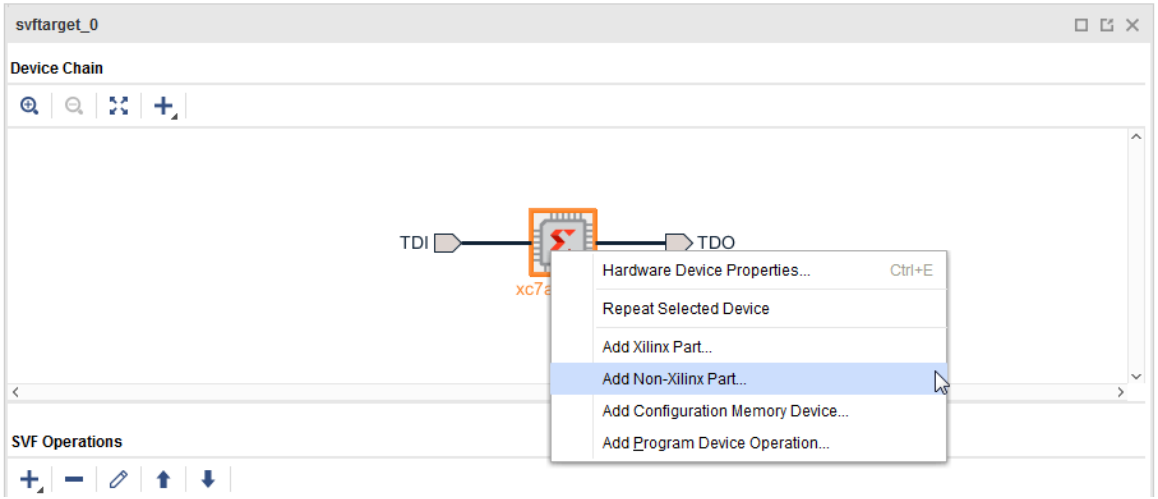
选中 AMD 器件并单击“OK”（确定）后，即可将此 AMD 器件添加到 SVF 链中，如下图所示。

图 65：**SVF 链中的 AMD 器件**



您还可以通过右键单击 SVF 链并单击“Add Non-Xilinx Part”（添加非 AMD 部件）来向 SVF 器件链添加非 AMD 部件，如下所示：

图 66：**添加非 AMD 部件**



这样会打开“Add Non-Xilinx Device”（添加非 AMD 器件）对话框，如下所示：

图 67：**“Add Non-Xilinx Device”对话框**

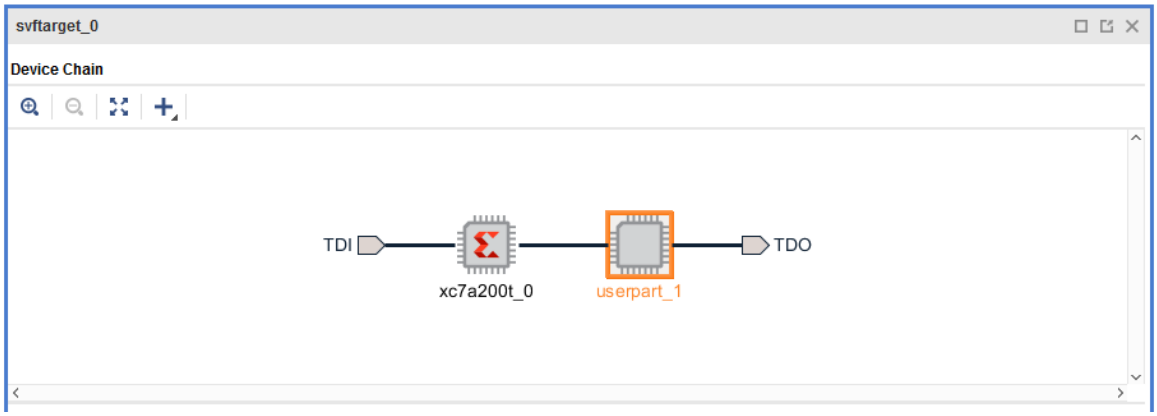
填充此对话框，如下所述：

- “Part Name”（部件名称）可填充您选择的任意部件名称。
- “ID code”（ID 代码）是十六进制值，表示有效的器件 ID 代码。
- “IR length”（IR 长度）是十进制数值，表示指令寄存器长度。
- “Mask”（掩码）是十六进制比特掩码值。

注释： “ID code”、“IR Length”和“Mask”值通常由硅片供应商通过器件 BSDL 文件来提供。

单击“OK”即可将非 AMD 部件添加到 SVF 器件链中。

图 68：**SVF 链中的非 AMD 器件**



使用命令行

要在 Vivado IDE 中使用 Vivado Tcl 模式或 Tcl 控制台来创建 JTAG 链，请在已打开的 SVF 目标上按顺序执行 `create_hw_device` 操作。例如，要先添加 xcku9p 部件，然后添加 xcvu095 部件，请执行以下步骤：

```
current_hw_target my_svf_target open_hw_target create_hw_device -part
xcku9p create_hw_device -part xcvu095 refresh_hw_target get_hw_devices
```

在此示例中，如果已创建并已打开 SVF，则可跳过前 2 个步骤。此示例中的 create_hw_device 命令用于定义 JTAG 链中的器件，从该链上的首个器件开始。

注释： create_hw_device 命令仅在已打开的 SVF 硬件目标上创建器件。

要将用户定义的器件添加到此链中，请使用 -part options 随部件类型名称一起添加 -idcode、-irlength，和 -mask 值。例如，如果部件名为“my_part”、“JTAG idcode”为 1234567、“ir length”为 8 且 mask 为 ffffffff，请按如下所示创建器件：

```
open_hw_target [current_hw_target] create_hw_device -idcode 01234567 -
irlength 8 -mask ffffffff -part my_part # print IR length for user defined
devices puts [get_property IR_LENGTH [lindex [get_hw_devices -filter {PART
== my_part}] 0]] puts $idcode_hex close_hw_target
```

注释： create_hw_device 的 idcode 应为有效的器件 ID 代码。ID 代码值和 IR 长度通常由硅片供应商通过器件 BSDL 文件来提供。

要查看目标及其器件的报告，请运行 report_hw_targets 命令。此报告可显示系统中所有活动目标的详细信息。此报告可用于获取服务器、目标和器件的属性，如下所示：

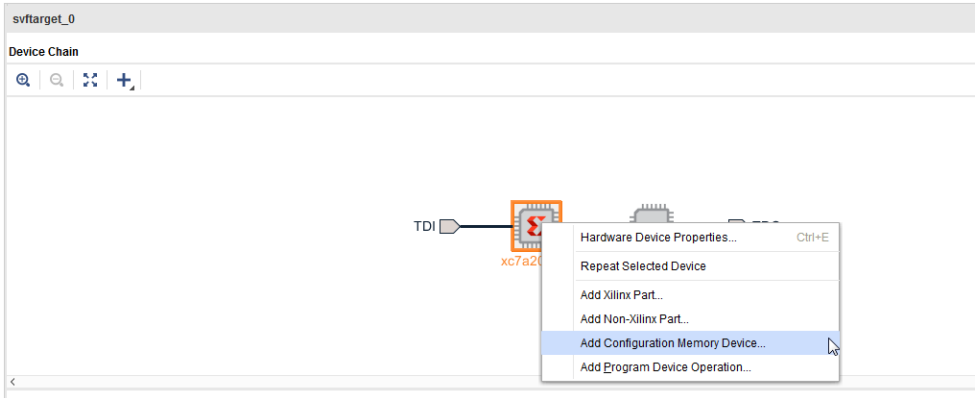
```
report_hw_targets INFO: Server Property Information: localhost:3121
    CLASS: hw_server     HOST: localhost     NAME: localhost:3121     PORT:
3121     SID: TCP:localhost:3121 INFO: Target Property Information:
localhost:3121/xilinx_tcf/Xilinx/my_svf_target     CLASS: hw_target
    DEVICE_COUNT: 3     HW_JTAG: 0     IS_OPENED: 1     MAX_DEVICE_COUNT:
32     NAME: localhost:3121/xilinx_tcf/Xilinx/my_svf_target     FREQUENCY:
10000000     TYPE: xilinx_tcf     TID: jsn-XNC-my_svf_target     UID:
Xilinx/my_svf_target     SVF: 1     Device: xcku9p_0     Device: xcvu095_1
     Device: my_part_2
```

向 AMD 器件添加配置存储器部件

使用 Vivado IDE

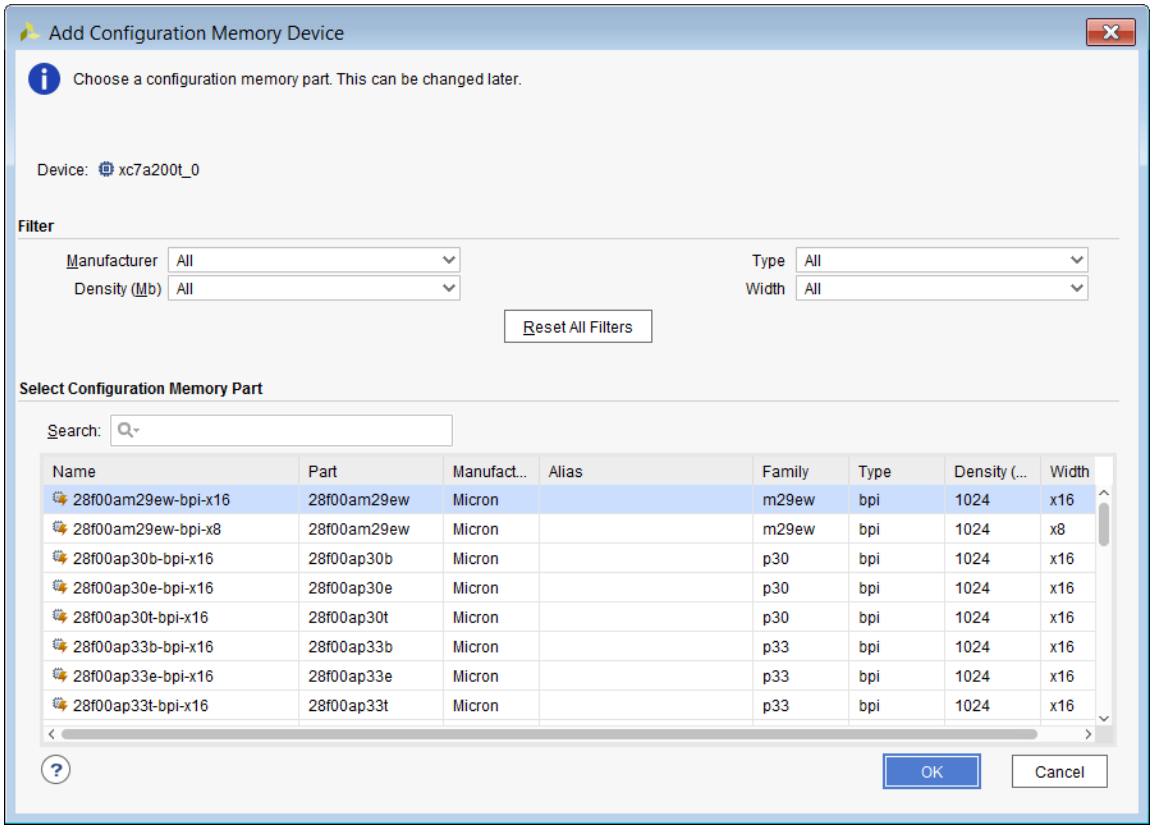
右键单击 SVF 链中的 AMD 器件部件时，可以选择创建配置存储器器件，并将配置存储器器件与该器件关联。

图 69：**添加配置存储器器件**



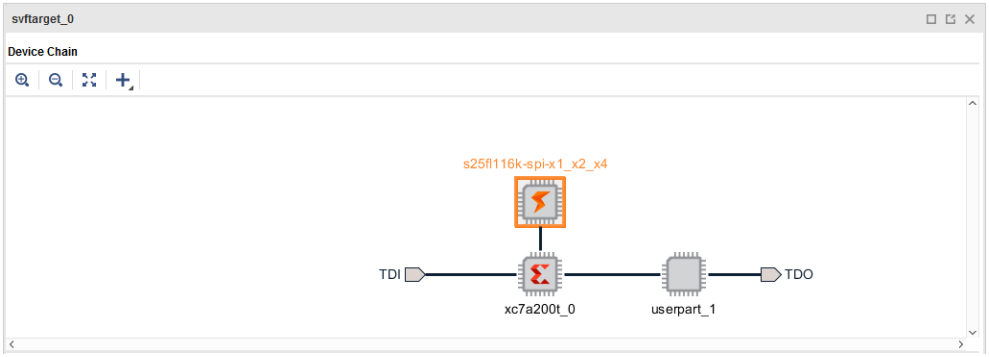
这样会打开“Add Configuration Memory Device”（添加配置存储器器件）对话框，如下所示：

图 70：**“Add Configuration Memory Device”对话框**



选择相应的存储器器件，然后单击“OK”（确定）。这样会将此器件与 AMD 器件关联，并显示在 SVF 器件链中，如下所示：

图 71：**SVF 链中的配置存储器器件**



使用命令行

要在 Vivado IDE 中使用 Vivado Tcl 模式或 Tcl 控制台来创建并关联配置存储器器件，请使用 `create_hw_cfgmem` Tcl 命令，如下所示：

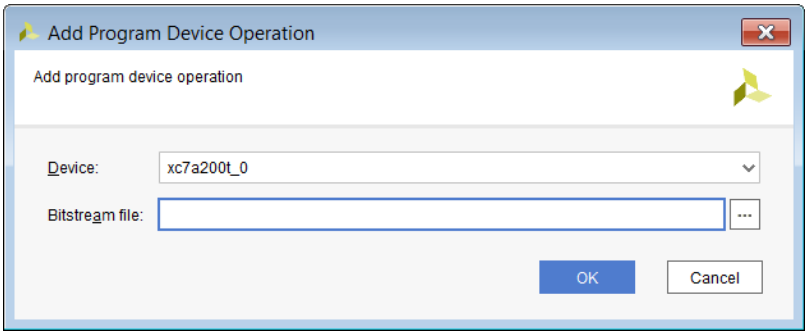
```
create_hw_cfgmem -hw_device [lindex [get_hw_devices xc7a200t_0] 0] [lindex
[get_cfgmem_parts {s25fl116k-spi-x1_x2_x4}] 0]
```

有关 SVF 链的操作

按正确顺序创建反映所有器件及其配置存储器的 SVF 链之后，即可开始向 SVF 链中的器件添加烧录操作。

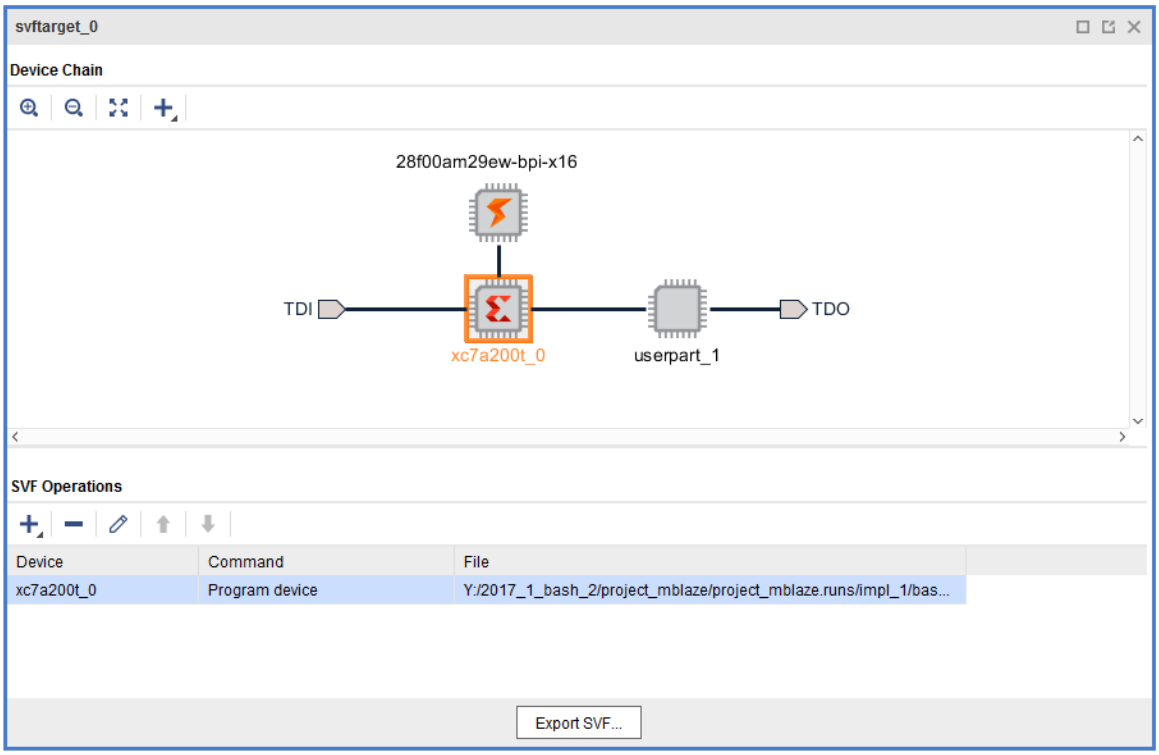
例如，您可右键单击链中的 AMD a200t 器件，然后选择“Add Program Device Operation”（添加器件烧录操作）以启动“Add Program Device Operation”对话框，如下所示。指定比特流文件，以便将其用于烧录器件。

图 72： **“Add Program Device Operation”对话框**



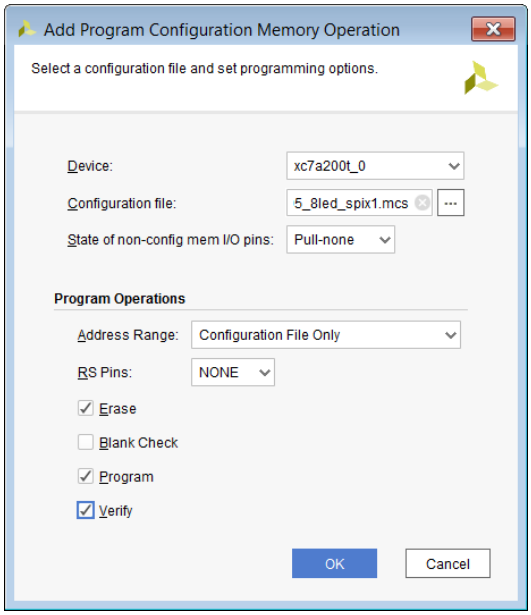
单击“OK”后，就会在“SVF Operations”（SVF 操作）窗口底部列出器件烧录操作。

图 73：**“SVF Operations”窗口**



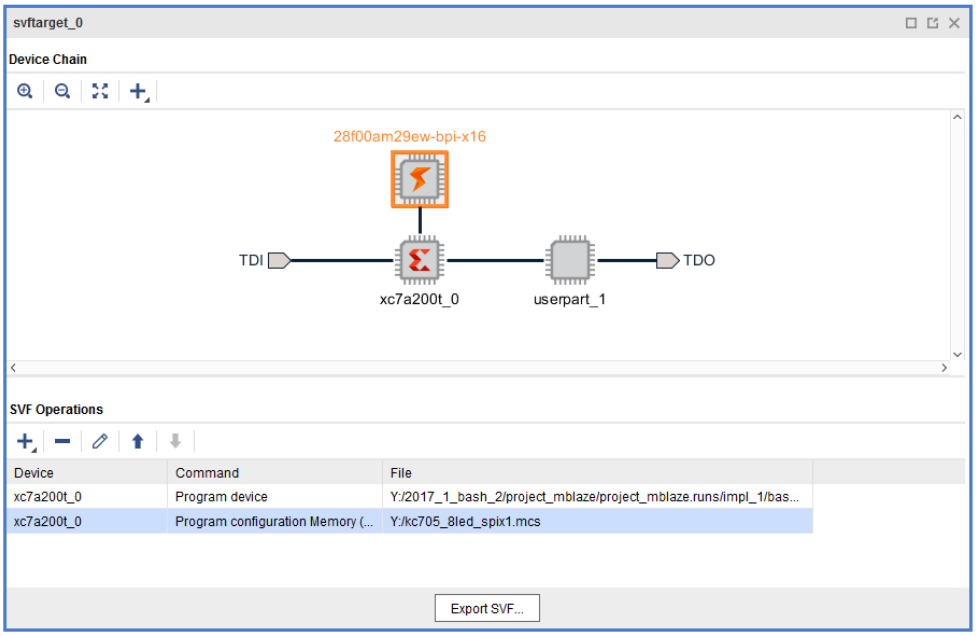
同样，您可通过右键单击存储器器件并选择“Add Program Configuration Memory”（添加烧录配置存储器）以启动“Add Program Configuration Memory”对话框来对配置存储器器件进行烧录，如下所示。指定配置文件，以便将其用于对存储器器件进行烧录。您也可以为存储器器件选择其他烧录选项，例如“Erase”（擦除）、“Blankcheck”（空白检查）和“Verify”（验证）。

图 74：**“Add Program Configuration Memory”对话框**



单击“OK”后，就会在“SVF Operations”窗口底部列出配置存储器器件烧录操作。

图 75： **“SVF Operations”窗口中的“Configuration Memory Device”**

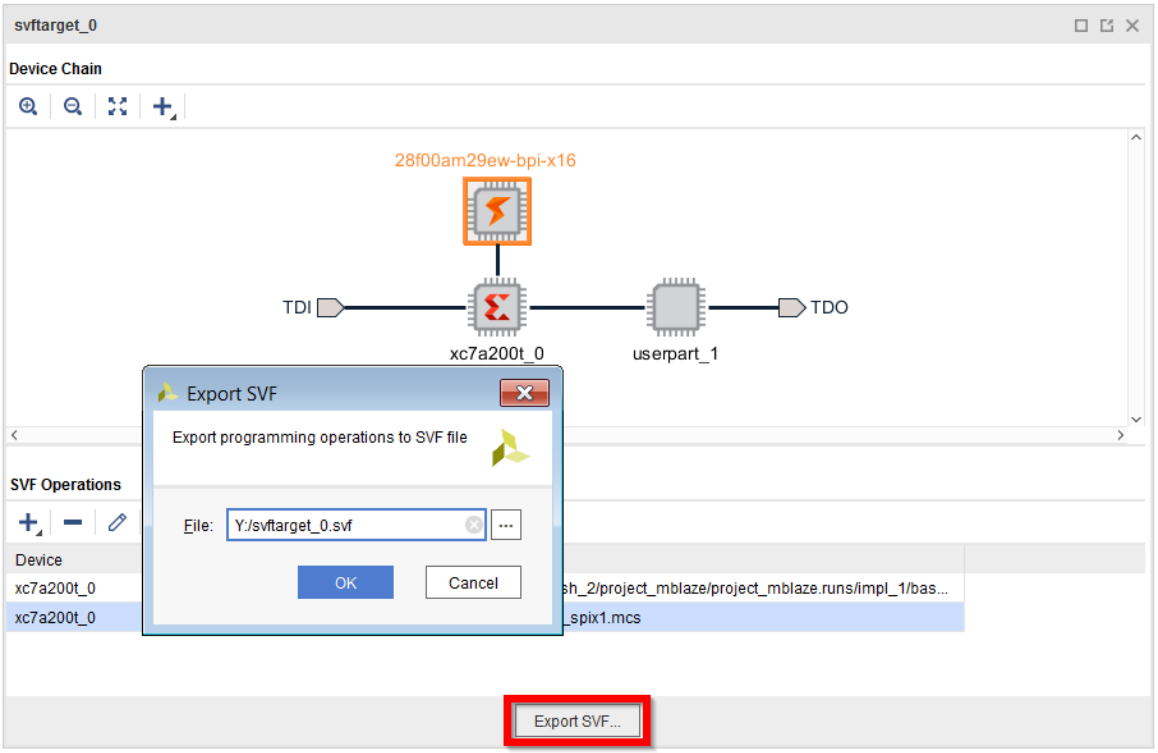


写入 SVF 文件

使用 Vivado IDE

通过单击位于“SVF Operations”（SVF 操作）窗口底部的“Export SVF”（导出 SVF），即可将 SVF 链设置及其操作保存至文件中，如下所示。

图 76：**导出 SVF 链设置**



重要提示！通过指定在先前流程运行中使用 Vivado 硬件管理器创建的 SVF 文件，即可重新创建现有 SVF 链。Vivado IDE 会将 SVF 链的规格保存到文件中，以便在回读时可重新创建该 SVF 链。

使用命令行

要使用 Vivado Tcl 模式或者 Tcl 控制台来编写 SVF 文件，请在 Vivado IDE 中使用 `write_hw_svf` 命令。

这样会在临时文件中捕获 SVF 链、直接 FPGA 和间接闪存烧录操作。调用 `write_hw_svf` 命令时，临时文件将改为传递给该命令的文件名。调用 `write_hw_svf` 命令后，临时文件将复位，并在 SVF 文件序列开头处添加 1 项后续烧录操作。

以下代码段显示了用于创建名为 `my_xcku9p.svf` 的文件的 Tcl 命令（包括对 xcku9p 器件进行直接烧录）：

```
create_hw_target my_svf_target open_hw_target set device0 [create_hw_device
-part xcku9p] set_property PROGRAM.FILE {my_xcku9p.bit} $device0
program_hw_devices $device0 write_hw_svf my_xcku9p.svf close_hw_target
```

在此样本代码中，xcku9p 器件是使用 `create_hw_device` 命令创建的，其返回值设置为名为 `device0` 的临时变量。将 `PROGRAM.FILE` 属性设为 `my_xcku9p.bit` 文件时，此临时值将用于引用对象。下一步，将使用 `device0` 引用来调用 `program_hw_device` 命令。运行此 `program_hw_device` 命令时，它会通过必要的 SVF 操作来创建临时 SVF 文件，用于对 xcku9p 上的 `my_xcku9p.bit` 文件执行烧录。最后，`write_hw_svf` 命令会将此临时文件移至最终目标 `myxcku9p.svf`。此时，SVF 文件创建进程即告完成，并且可关闭目标。

提示：编写 STAPL 文件时，应首先为 JTAG 链创建所有器件，然后再执行烧录操作。如果在执行烧录命令间交织执行了 `create_hw_device` 命令，那么生成的输出 SVF 文件会包含 2 条不同的序列链。

错误的 SVF 文件创建步骤示例：

```
create_hw_target my_svf_target open_hw_target set device0
[create_hw_device -part xcku9p] set_property PROGRAM.FILE
{my_xcku9p1.bit} $device0 # this program command will produce SVF
instructions #  which account for only device0 in chain
program_hw_devices $device0 set device1 [create_hw_device -part xcku9p]
set_property PROGRAM.FILE {my_xcku9p2.bit} $device1 # this program
command will produce SVF instructions #  which account for device0 and
device1 in chain program_hw_devices $device1 write_hw_svf
my_bad_xcku9p.svf close_hw_target
```

第一条烧录命令仅捕获包含首个器件的链定义。第二条烧录命令在写出 SVF 指令时会包含链中的 2 个器件。因此如果您尝试在含 2 个器件的链上运行此 SVF 文件，则第一项烧录操作将失败，因为活动链会收到 2 个器件，而非此命令期望的 1 个器件。

要纠正此问题，请首先运行 `create_hw_device` 命令。当链完成定义后，请按如下所示执行烧录操作：

正确的 SVF 文件创建步骤示例

```
create_hw_target my_svf_target open_hw_target # create device chain first
set device0 [create_hw_device -part xcku9p] set device1 [create_hw_device -
part xcku9p] # program device0 set_property PROGRAM.FILE {my_xcku9p1.bit}
$device0 program_hw_devices $device0 # program device1 set_property
PROGRAM.FILE {my_xcku9p2.bit} $device1 program_hw_devices $device1
write_hw_svf my_good_xcku9p.svf close_hw_target
```

执行 SVF 文件

创建 SVF 文件后，您可选择通过 Vivado IDE 来执行 SVF 文件。Vivado IDE 可以执行通过 SVF 生成功能所生成的 SVF 文件，主要用作为确认测试工具。`execute_hw_svf` 命令并非常用的 SVF 执行命令，请注意，只能使用通过 Vivado IDE 创建的 SVF 文件。

要运行 `svf` 命令，请在已打开并处于活动状态的目标上运行如下命令：

```
execute_hw_svf my_file.svf INFO: [Labtoolstcl 44-548] Creating JTAG TCL
script from SVF file INFO: [Labtoolstcl 44-549] Re-opening target in JTAG
mode INFO: [Labtoolstcl 44-551] Sourcing JTAG TCL script: my_file.tcl Pass:
SVF Execution completed with no errors INFO: [Labtoolstcl 44-550] Restoring
target to original mode INFO: [Labtoolstcl 44-570] Execute SVF completed
successfully
```

在本例中，执行的文件是 `my_file.svf`。在执行流程中，输入 SVF 文件可通过 HW_JTAG Tcl 操作转换为临时文件。创建此 Tcl 代码后，将使用此文件来执行转换后的 SVF 指令。要查看 JTAG_TCL 操作，可使用 `-verbose` 选项运行 `execute_hw_svf` 命令。该命令完成后，消息日志末尾会显示标明指令执行失败位置的出错消息或表示执行成功的“Pass”消息。

提示：Vivado 支持对小于 500 MB 的 SVF 文件执行 SVF。要对大小超过 500 MB 的 SVF 文件执行操作，请使用第三方 SVF 播放器。

第 11 章

设计调试

对 FPGA 或自适应 SoC 设计进行调试是一个多步骤迭代式进程。与大多数复杂问题的处理方式一样，最好先将 FPGA 或自适应 SoC 设计调试进程细分为多个小部分，以便集中精力使设计中的每一小部分能逐一正常运行，而不是尝试一次性让整个设计都能正常运行。举例来说，逐一添加每个模块，使其在整个设计环境内都正常运行，以此方式迭代完成整个设计流程，这正是经过验证的设计和调试方法论示例之一。您可通过混用以下任意设计流程阶段来使用此设计与调试方法论：

- RTL 级别设计仿真
- 实现后设计仿真
- 系统内调试

RTL 级别设计仿真

在仿真验证进程中，可对设计功能进行调试。AMD 在 AMD Vivado™ IDE 中提供了完整的设计仿真功能。AMD Vivado™ 设计仿真器可用于执行设计的 RTL 仿真。在 RTL 级别仿真环境内执行设计调试的优势包括整个器件完整可见并且能够通过设计/调试周期进行快速迭代。使用 RTL 级别仿真执行设计调试的局限性包括难以在合理时间范围内对较大型的设计进行仿真，以及难以对实际系统环境进行准确仿真。如需了解有关使用 AMD Vivado™ 仿真器的更多信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》(UG900)。

实现后设计仿真

Vivado 仿真器还可用于对实现后的设计进行仿真。使用 Vivado 仿真器对实现后设计进行调试的好处包括能够获得时序准确的设计模型。执行实现后设计仿真的局限性如上一章所述，包括：运行时间较长以及系统模型准确性欠佳。

系统内逻辑设计调试

Vivado Design Suite 还包含逻辑分析功能，支持您对 FPGA 或自适应 SoC 中实现后的设计执行系统内调试。在系统内调试设计的好处包括：能够在以系统级速度运行的实际系统环境内对实现后时序准确的设计进行调试。系统内调试的局限性包括：相比于仿真模型，调试信号可见度较低，设计、实现或调试迭代耗时可能较长（取决于设计的规模和复杂性）。

一般而言，Vivado 工具可提供多种不同方法用于设计调试。您可以根据自身需求使用其中一种或多种方法来调试设计。系统内逻辑设计调试流程主要围绕 Vivado Design Suite 的系统内逻辑调试功能展开。

相关信息

系统内逻辑设计调试流程

系统内串行 I/O 设计调试

为启用系统内串行 I/O 确认和调试，Vivado Design Suite 包含串行 I/O 分析功能。这样您即可在基于 FPGA 的系统内对自己的高速串行 I/O 链路进行测量和优化。Vivado Serial I/O Analyzer 功能旨在帮助您解决各种系统内调试和确认问题，从简单的时钟设置和连接问题到复杂的裕度分析和通道优化问题都不在话下。使用 Vivado Serial I/O Analyzer 相比于其他外部检测方法的优势在于，您测量的是对接收到的信号应用接收器均衡后的信号质量。这样可确保在发射到接收通道中的最优点执行测量，从而确保获取真实准确的数据。

Vivado 工具不仅提供了用于实践千兆位收发器端点的设计的生成方法，还提供了运行时软件用于执行测量并帮助您对高速串行 I/O 通道进行优化。“串行 I/O 硬件调试流程”可逐步指导您完成生成 IBERT 设计的进程。“在硬件中调试串行 I/O 设计”可逐步指导您使用运行时 Vivado Serial I/O Analyzer 功能。

相关信息

串行 I/O 硬件调试流程
在硬件中调试串行 I/O 设计

第 12 章

系统内逻辑设计调试流程

AMD Vivado™ 工具提供了诸多功能，用于在真实硬件器件中调试系统内设计。系统内调试流程包含 3 个不同阶段：

1. **探测阶段**：确定设计中要探测的信号和探测的方法。
2. **实现阶段**：完成设计实现，包括连接到所探测的信号线上的其他调试 IP。
3. **分析阶段**：与设计中包含的调试 IP 进行交互，以便对功能问题进行调试和验证。

此系统内调试流程按设计应使用上一章节内所述迭代设计/调试流程来运行。如果您选择使用系统内调试流程，那么建议在设计周期中尽早使您的部分设计能够在硬件中运行。本章其余部分描述了系统内调试流程的 3 个阶段，以及如何使用 Vivado 逻辑调试功能来使设计尽早在硬件内正常工作。

通过设计探测来执行系统内调试

系统内调试流程的探测阶段分为 2 个步骤：

1. 识别要探测的信号或信号线。
2. 决定将调试核添加到设计中的方式。

在许多情况下，您所决定的要探测的信号及其探测方式会导致这些信号彼此之间相互影响。最好首先决定是要手动将调试 IP 组件实例添加到设计源代码（称为 HDL 例化探测流程），还是要使用 Vivado 工具来将调试核自动插入综合后的网表（称为网表插入探测流程）。下表描述了不同调试方法的部分利弊取舍。

表 4：**调试策略**

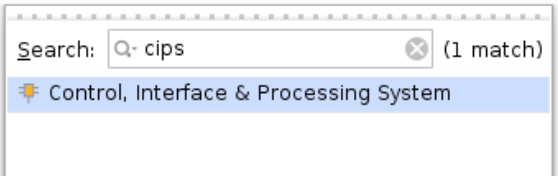
调试目标	推荐的调试烧录流程
识别 HDL 源代码中的调试信号，同时保留稍后在流程中启用/禁用调试的灵活性。	使用 mark_debug 属性来标记要在 HDL 中进行调试的信号。 使用“Set up Debug”Wizard（设置调试向导）来指导您完成“Netlist Insertion”（网表插入）探测流程。
识别已综合的设计网表中的调试信号线，无需修改 HDL 源代码。	使用“Mark Debug”（标记调试）右键单击菜单选项从已综合的设计网表中选择要调试的信号线。 使用“Set up Debug”Wizard（设置调试向导）来指导您完成“Netlist Insertion”（网表插入）探测流程。
使用 Tcl 命令自动执行调试探测流程。	使用 `set_property` Tcl 命令在调试信号线上设置 mark_debug 属性。 使用“网表插入”探测流程的 Tcl 命令来创建调试核，并将其连接到调试信号线。
在 HDL 源中将信号显式连接到 ILA 调试核实例。	识别要调试的 HDL 信号。 使用“HDL 例化”探测流程来生成并例化 Integrated Logic Analyzer (ILA) 核，并将其连接到设计中的调试信号。

Versal 系统内调试

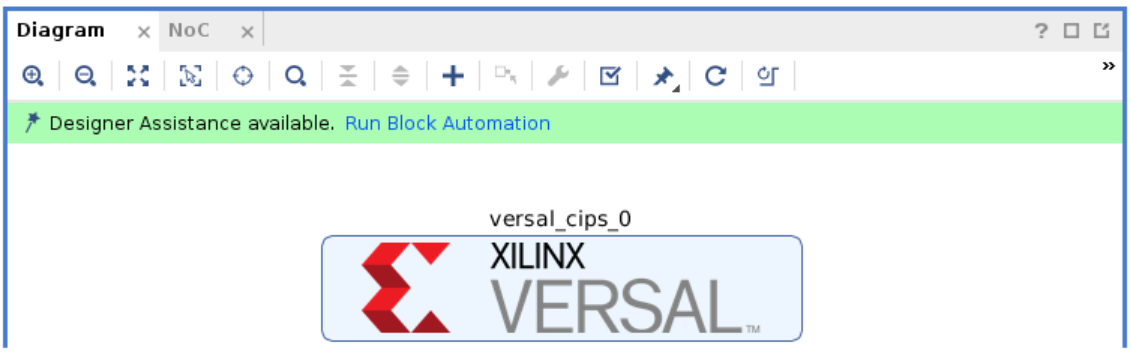
虽然 AMD Versal™ 自适应 SoC 架构有别于先前 FPGA 架构，并且使用不同的调试 IP 和基础架构来连接系统内调试核，但调试流程与先前 FPGA 架构存在诸多相似之处。本章中详细讲解了调试流程之间的差异。

添加 Control, Interfaces, and Processing System (CIPS)

1. 如果设计不含块设计，请单击 Flow Navigator 的“IP integrator”类别下的“Create Block Design”（创建块设计）以创建块设计。
2. 单击“+”以将新的 IP 添加到 IP integrator 画布上，并输入 `cips` 以搜索 CIPS IP（如下图所示）。找到后，请双击以将其添加到 IP integrator 画布中。



3. 添加后，将在工具栏顶部附近显示一个绿条，以指示设计辅助功能可供使用。除非需要配置其他选项，否则无需运行块自动化设置。



4. 单击相应的按钮以确认块设计，然后单击相应的按钮以保存块设计。
5. 返回至“Project Manager”（工程管理器）、右键单击新创建的块设计并单击“Create HDL Wrapper”（创建 HDL 封装文件），为块设计生成 HDL 封装文件。
6. 创建后，请确保此块设计已例化为设计的一部分。
7. 使用“Netlist Insertion Debug Probing Flow”（网表插入调试探测流程）、“HDL Instantiation Debug Probing Flow”（HDL 例化调试探测流程）或“IP integrator Debug Flow”（IP integrator 调试流程）继续执行操作。如果设计包含任何调试核，那么在执行 `opt_design` 期间会将 AXI4 Debug Hub 自动添加到网表中，并且会自动连接调试核。

   注释：默认情况下，除非添加了需要 I/O 的额外 IP，否则此块设计将不含任何输入或输出端口。

注释：对于大部分应用，无需手动将例化的调试核连接到 AXI4 Debug Hub。执行 `opt_design` 期间，会将 AXI4 Debug Hub 和 NoC 的实例插入网表，并且会在调试核、Debug Hub 与 CIPS 核之间自动建立连接。

注释：如需获取有关使用 CIPS 核执行设计创建、仿真和调试的更多信息，请参阅《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)。

AXI4 Debug Hub

Versal 器件使用的是 AXI4 Debug Hub，它与先前架构中使用的 Debug Hub 类似，并可在 Versal Control, Interfaces, and Processing System (CIPS) IP 与设计中的调试核之间提供连接。在先前架构中，Vivado 会自动插入该核，无需用户干预。您也可以手动例化 AXI4 Debug Hub。建议仅当使用 DFX 时或者当设计的寻址方案要求设置 AXI4 Debug Hub 的手动地址时，才执行手动例化。

AXI4-Stream ILA

Versal 器件使用包含 ILA 和 System-ILA 功能的 AXI4-Stream ILA。AXI4-Stream ILA 还为走线存储器提供了 2 个选项：块 RAM (BRAM) 和 UltraRAM (URAM)。

AXI4 Debug Hub 连接

要使用 AMD Vivado™ 调试核，设计必须包含 AXI4 Debug Hub。AXI4 Debug Hub 可用于将 CIPS 的 AXI4 接口与 AXI4-Stream 接口相连。此接口可连接到 Vivado 调试核，其中包含以下类型的核：

- AXI4-Stream Integrated Logic Analyzer (AXIS-ILA)
- AXI4-Stream Virtual Input/Output (AXIS-VIO)
- PCI Express® Link Debugger

表 5：**AXI4 Debug Hub 自动插入**

AXI4 Debug Hub 连接	调试流程
AXI4 Debug Hub 综合后网表自动插入和连接。	建议将此方法用于大部分用例，因为此方法可提供最大的灵活性。 1. 在 `opt_design` 期间，Vivado 调试流程会检测设计是否包含任何需要 AXI4 Debug Hub 连接的调试核。Vivado 调试流程还会检测设计是否包含 Control, Interfaces, and Processing System (CIPS) IP 实例。 2. Vivado 调试流程会在综合后的网表中插入 1 个 AXI4 Debug Hub 实例，并将其自动连接到设计中所用的调试核。 **注释：**对于使用 Dynamic Function eXchange (DFX) 的设计，无法使用此方法。

表 5：**AXI4 Debug Hub 自动插入** (续)

AXI4 Debug Hub 连接	调试流程
AXI4 Debug Hub 手动例化，综合后网表调试核自动连接。	在下列情况下应使用此方法：手动分配 AXI4 Debug Hub 所用的地址，或者使用 Dynamic Function eXchange (DFX) 时。在此情况下，在设计中，应手动例化 AXI4 Debug Hub，并从 Control, Interfaces, and Processing System (CIPS) IP 连接到 AXI4 主接口。 1. 在 `opt_design` 期间，Vivado 调试流程会检测设计是否包含任何需要 AXI4 Debug Hub 连接的调试核。Vivado 调试流程还会检测设计是否包含 Control, Interfaces, and Processing System (CIPS) IP 实例。 2. Vivado 调试流程会定位手动添加的 AXI4 Debug Hub。此 AXI4 Debug Hub 实例会替换为 AXI4 Debug Hub，其中配置有合适数量的 AXI4-Stream 接口，以供连接到设计中使用的每个调试核。 **注释：**在上述步骤中，Vivado 调试流程所替换的 AXI4 Debug Hub 会保留用户指定的地址和属性。
AXI4 Debug Hub 手动例化，调试核手动连接。	如需手动定义 AXI4 Debug Hub、CIPS 和设计中的所有调试核之间的所有连接，则应使用此方法。对于使用 Dynamic Function eXchange (DFX) 的设计，同样可采用此方法。 1. 构建设计时，会将 1 个或多个 AXI4 Debug Hub 实例添加到设计中，并连接到 CIPS IP 上的相应 AXI4 主接口。 2. AXI4 Debug Hub 应自定义为所含 AXI4-Stream 接口数量与设计中的调试核的数量完全相同。 3. 设计中的每个调试核都应具有相应的选项，用于为开启的手动连接启用 AXI4-Stream 端口。 4. 用户应自行负责将每个调试核的 AXI4-Stream 主接口和从接口连接到 AXI4 Debug Hub 上的对应从接口和主接口。

注释：手动连接 AXI4 Debug Hub 时，建议使用 PMC NoC 接口，因为这是到调试包控制器的专用路径。也可以使用 FPD 或 LPD 接口，不支持扩展地址范围 0x004_0000_0000 (8G) 和 0x400_0000_0000 (1T)。

手动连接 Versal 调试核

Versal AXI4 Debug Hub IP 和调试核（例如，AXI4-Stream ILA、AXI4-Stream VIO）可提供相应的选项，用于手动定义调试核与 AXI4 Debug Hub 之间的连接。大部分设计都无需使用该选项。

要在 AXI4 Debug Hub 与调试核之间启用手动连接，请执行以下操作：

1. 生成并例化 AXI4 Debug Hub IP 实例，然后使用 PMC 将其连接到设计中的 Control, Interfaces, and Processing System (CIPS) IP。
2. 自定义 AXI4 Debug Hub IP，将“Number of Debug Cores”（调试核数）设置为要手动连接的调试核的精确数量。对于每个调试核，在 AXI4 Debug Hub IP 上都会显示 1 个 AXI4-Stream 主接口和 1 个从接口。

   注释：AXI4 Debug Hub 上未连接的 AXI4-Stream 端口可能导致执行 `opt_design` 期间发生错误。
3. 在对应要手动连接的调试核的 IP 自定义接口中，选中“Enable AXI4-Stream Interfaces for Manual Connection to AXI Debug Hub”（启用 AXI4-Stream 接口以便手动连接到 AXI Debug Hub）。这样就会在调试核上显示 AXI4-Stream 主端口和从端口。

4. 将调试核上的每个 AXI4-Stream 主端口和从端口连接到 AXI4 Debug Hub 上的相应端口。调试核也包含 `aclk` 和 `aresetn` 端口，这 2 个端口应连接到与 AXI4 Debug Hub 相连的相同时钟和复位端口。

使用网表插入调试探测流程

在 Vivado 工具中插入调试核的过程以分层方式来演示，以满足不同 Vivado 用户组的不同需求：

- 最高层提供了一套简单向导，用于根据选定待调试的一组信号线来自动创建并配置 Integrated Logic Analyzer (ILA) 核。
- 下一层是“Debug”（调试）主窗口，此窗口支持控制个别调试核、端口及其属性。从“Layout Selector”（布局选择器）或“Layout”（布局）菜单中选择“Debug”（调试）布局以打开“Synthesized Design”（已综合的设计）时即可显示“Debug”（调试）窗口，或者也可以使用“Window”→“Debug”（窗口 > 调试）来直接打开此窗口。
- 最底层是一组 Tcl XDC 调试命令，您可在 XDC 约束文件中手动输入这些命令，或者也可以将其作为 Tcl 脚本来进行重复运行。

您也可将多种模式组合使用，以插入和自定义调试核。

标记要调试的 HDL 信号

您可在综合前使用 mark_debug 约束在 HDL 源代码级别识别要调试的信号。在“Debug”（调试）窗口中的“Unassigned Debug Nets”（未分配的调试信号线）文件夹下，会自动列出对应于 HDL 中标记调试的信号的信号线。

注释：在“Debug”窗口中，“Debug Nets”（调试信号线）视图是主要围绕信号线的视图，其中显示了您选中调试的信号线。“Debug Cores”（调试核）视图则是主要围绕核的视图，您可在其中查看和设置核属性。

标记调试的信号线的过程取决于您当前正在处理基于 RTL 源代码的工程还是基于已综合的网表的工程。对于基于 RTL 网表的工程：

- 通过使用 Vivado 综合功能，您就可以有选择性地使用 VHDL 和 Verilog 源文件中的 mark_debug 约束来标记要调试的 HDL 信号。mark_debug 约束的有效值为“TRUE”或“FALSE”。Vivado 综合功能不支持“SOFT”值。

对于基于已综合的网表的工程：

- 通过使用 Synopsys® Synplify® 综合工具，您就可以有选择性地使用 VHDL 或 Verilog 中的 mark_debug 和 syn_keep 约束来标记要调试的信号线，或者也可以使用 Synopsys 设计约束 (SDC) 文件中的 mark_debug 约束来进行标记。Synplify 不支持“SOFT”值，因为此行为由 syn_keep 属性来控制。
- 通过使用 Mentor Graphics® Precision® 综合工具，您就可以有选择性地使用 VHDL 或 Verilog 中的 mark_debug 约束来标记要调试的信号线。

以下小节提供了适用于 Vivado 综合、XST、Synplify 和 Precision 源文件的语法示例。

图标和 ILA 核

- 空心绿色图标 表示已设置 MARK_DEBUG 属性但并未连接到任何 ILA 核的信号线。
- 实心绿色图标 表示已设置 MARK_DEBUG 属性且已连接到 ILA 核的信号线。
- 黄色图标 表示信号线上未设置 MARK_DEBUG，但此信号线已连接到 ILA 核。

Vivado 综合 mark_debug 语法示例

以下是使用 Vivado 综合时的 VHDL 和 Verilog 语法示例。

- VHDL 语法示例

```
attribute mark_debug : string; attribute mark_debug of char_fifo_dout:
signal is "true";
```

- Verilog 语法示例

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

Synplify mark_debug 语法示例

以下是适用于 VHDL、Verilog 和 SDC 的 Synplify 语法示例。

- VHDL 语法示例

```
attribute syn_keep : boolean; attribute mark_debug : string; attribute
syn_keep of char_fifo_dout: signal is true; attribute mark_debug of
char_fifo_dout: signal is "true";
```

- Verilog 语法示例

```
(* syn_keep = "true", mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

- SDC 语法示例

```
define_attribute {n:char_fifo_din[*]} {mark_debug} {"true"}
define_attribute {n:char_fifo_din[*]} {syn_keep} {"true"}
```

重要提示！SDC 源代码中的信号线名称必须使用“n:”限定符作为前缀。

注释：Synopsys 设计约束 (SDC) 是业内普遍接受的标准，用于将设计意图告知工具，主要用于时序分析。请单击此链接并注册 TAP-in 程序，即可获取 Synopsys 所提供的 SDC 规范的参考副本。

Precision mark_debug 语法示例

以下是使用 Precision 时的 VHDL 和 Verilog 语法示例。

- VHDL 语法示例

```
attribute mark_debug : string; attribute mark_debug of char_fifo_dout:
signal is "true";
```

- Verilog 语法示例

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

设计综合

下一步是在 Vivado Design Suite 中单击“Run Synthesis”（运行综合）或者运行以下 Tcl 命令来对包含调试核的设计执行综合操作：

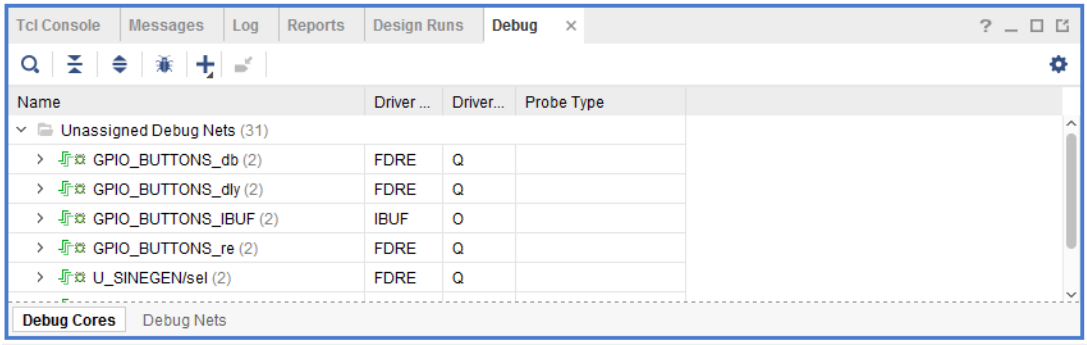
```
launch_runs synth_1 wait_on_run synth_1
```

您也可以使用 synth_design Tcl 命令来对设计执行综合。如需了解有关各种设计综合方法的更多详细信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

在已综合的设计中标记要调试的信号线

在 Flow Navigator 中单击“Open Synthesized Design”（打开已综合的设计），然后选择“Debug”（调试）窗口布局以查看“Debug”窗口。这样会在“Debug”窗口中的“Unassigned Debug Nets”（未分配的调试信号线）文件夹下显示对应于带调试标记的 HDL 信号的所有信号线。

图 77：**Unassigned Debug Nets**



- 选中“Netlist”（网表）或“Schematic”（板级原理图）窗口等任一设计视图中的信号线，然后右键单击并选择“Mark Debug”（标记调试）选项。
- 选中任一设计视图中的信号线，将其拖放到“Unassigned Debug Nets”文件夹中。
- 在“Set up Debug”Wizard（设置调试向导）中使用信号线选择器，欲知详情，请参阅：使用“Set Up Debug”Wizard 来插入调试核。

相关信息

使用“Set Up Debug”Wizard 来插入调试核

使用“Set Up Debug”Wizard 来插入调试核

标记要调试的信号线 (net) 后，下一步是将其分配到调试核。Vivado Design Suite 提供了易于使用的“Set up Debug”Wizard（设置调试向导），以帮助逐步指导您完成自动创建调试核并将调试信号线分配到这些核的输入的整个进程。

要使用“Set up Debug”Wizard 来插入调试核，请执行以下操作：

1. （可选）使用未分配的信号线列表来选择一组信号线以供调试，或者直接选择要调试的信号线。
2. 从 Vivado Design Suite 主菜单中依次选择“Tools”→“Set up Debug”（工具 > 设置调试），或者单击 Flow Navigator 中“Synthesized Design”（已综合的设计）部分下的“Set up Debug”（设置调试）。
3. 单击“Next”（下一步）以转至“Specify Nets to Debug”（指定要调试的信号线）面板（请参阅下图）。

4. （可选）单击“Find Nets to Add”（查找要添加的信号线）以在表中添加更多信号线，或者移除现有信号线。您还可右键单击调试信号线并选择“Remove Nets”（移除信号线）以从表中移除信号线。

   重要提示！您还可在“Netlist”（网表）或其他窗口中选中信号线，然后将其拖到“Nets to Debug”（要调试的信号线）列表中。

5. 右键单击调试信号线，并选中“Select Clock Domain”（选择时钟域）以更改将用于对信号线上的值进行采样的时钟域。

   注释：“Set up Debug”Wizard 会尝试搜索路径中的同步元件，以便为调试信号线自动选择相应的时钟域。“Select Clock Domain”（选择时钟域）对话框可用于按需修改此选择，但请注意，表中的每个时钟域都会生成 1 个独立的 ILA 核实例。

   提示：请参阅《适用于 FPGA 和 SoC 的 UltraFast 设计方法指南》(UG949) 中的 ILA 核与时序注意事项，以获取有助于最大限度降低 ILA 核的时序影响的技巧。

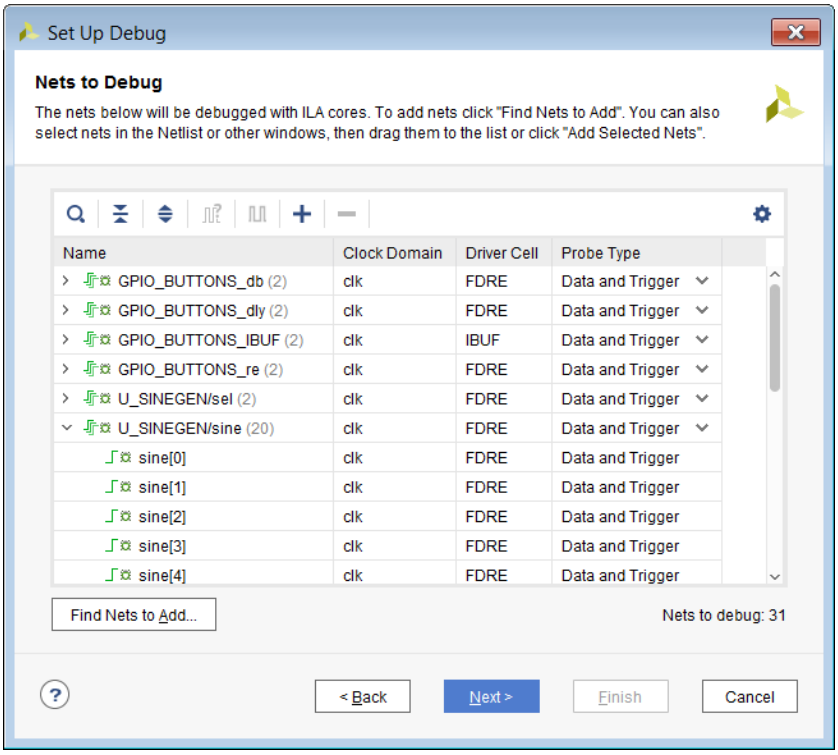
6. 对所选调试信号线满意后，请单击“Next”（下一步）。

   注释：“Set up Debug”Wizard 会为每个时钟域插入 1 个 ILA 核。为调试所选的信号线将自动分配到所插入的 ILA 核的探测端口。最后一个 Wizard 屏幕会显示核创建汇总信息，其中包括找到的时钟数以及要创建和/或移除的 ILA 核数。

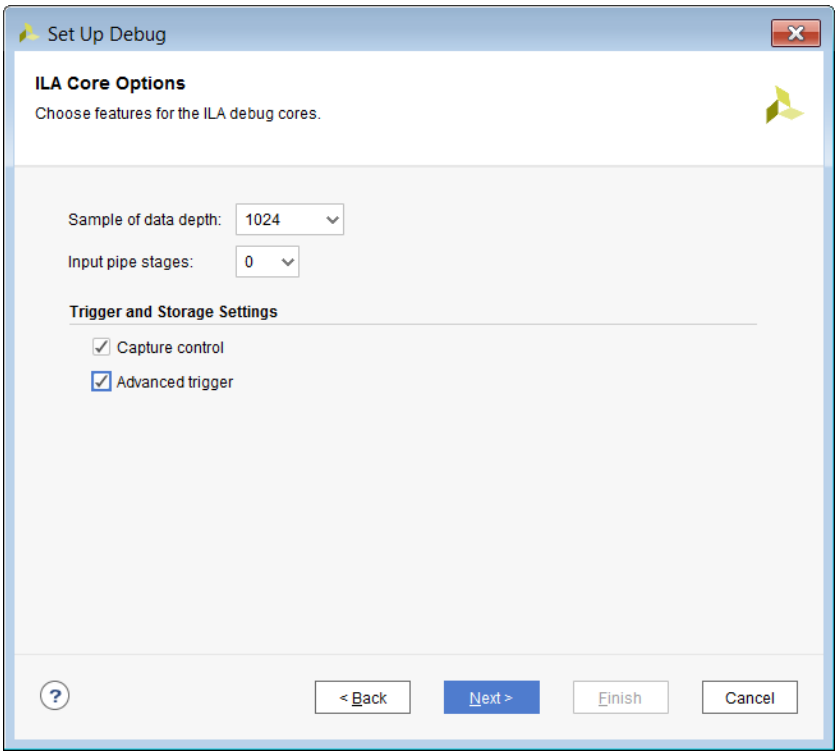
7. 如果要启用高级触发器模式或基本捕获模式，请使用对应的复选框。单击“Next”，移至最后一个面板。

   注释：如需了解有关在 Vivado 硬件管理器中使用的高级触发器模式和基本捕获模式功能的更多详细信息，请参阅“在硬件中调试逻辑设计”。

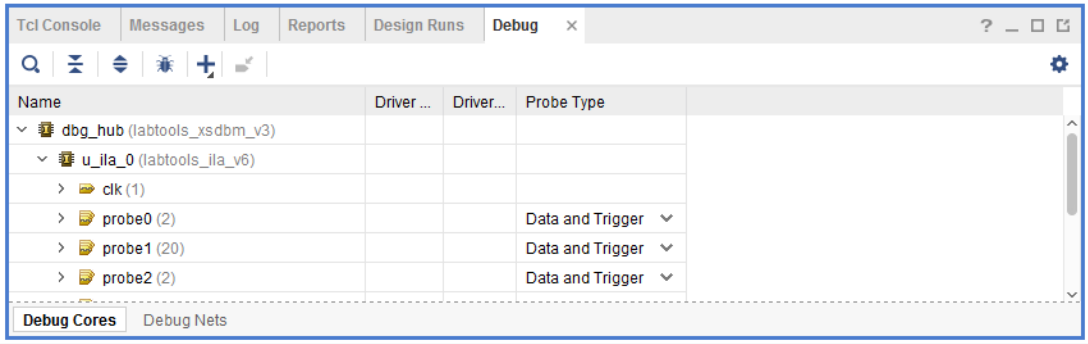
8. 如果您对结果感到满意，请单击“Finish”以将 ILA 核插入已综合的设计网表并在其中连接这些 ILA 核。



9. 配置 ILA 核常规选项，例如，ILA 数据深度 (C_DATA_DEPTH)、输入管道阶段数 (C_INPUT_PIPE_STAGES)、启用捕获控制功能 (C_EN_STRG_QUAL) 以及启用高级触发器功能 (C_ADV_TRIGGER)。请参阅“在调试核上修改属性”以获取有关这些选项的描述。



10. 现在，调试信号线已分配到 ILA 调试核，如下图所示。



相关信息

ILA 核与时序注意事项
在硬件中调试逻辑设计
在调试核上修改属性

使用调试窗口来添加和自定义调试核

相比于“Set up Debug”Wizard（设置调试向导），“Debug”（调试）窗口中的“Debug Cores”（调试核）选项卡可对 ILA 核和 Debug Hub 核插入提供更细化的控制。此窗口中提供的控制措施支持创建核、删除核、连接调试信号线和更改核参数。

“Debug”窗口的“Debug Cores”选项卡：

- 显示连接到 Debug Hub (`dbg_hub`) 核的调试核列表。

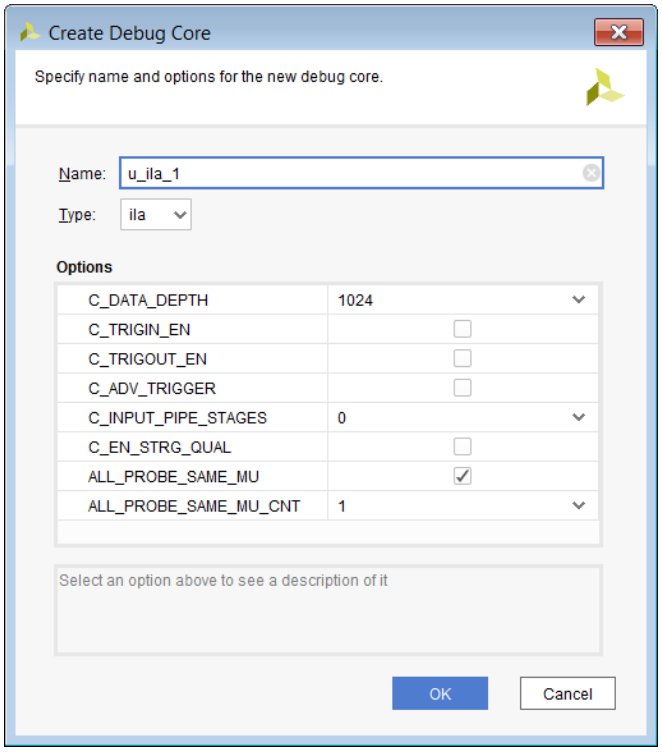
- 在窗口底部保持显示未分配的调试信号线列表。

您可使用弹出菜单或窗口顶部的工具栏按钮来操纵调试核与端口。

创建和移除调试核

要在“Debug”（调试）窗口中创建调试核，请单击“Create Debug Core”（创建调试核）。通过使用此接口即可更改父实例、调试核名称以及为核设置参数。要移除现有调试核，请在“Debug”窗口中右键单击该核，然后单击“Delete”（删除）。请参阅“在调试核上修改属性”以获取“Create Debug Core”对话框中的 ILA 核选项的描述。

图 78：**创建新的调试核**



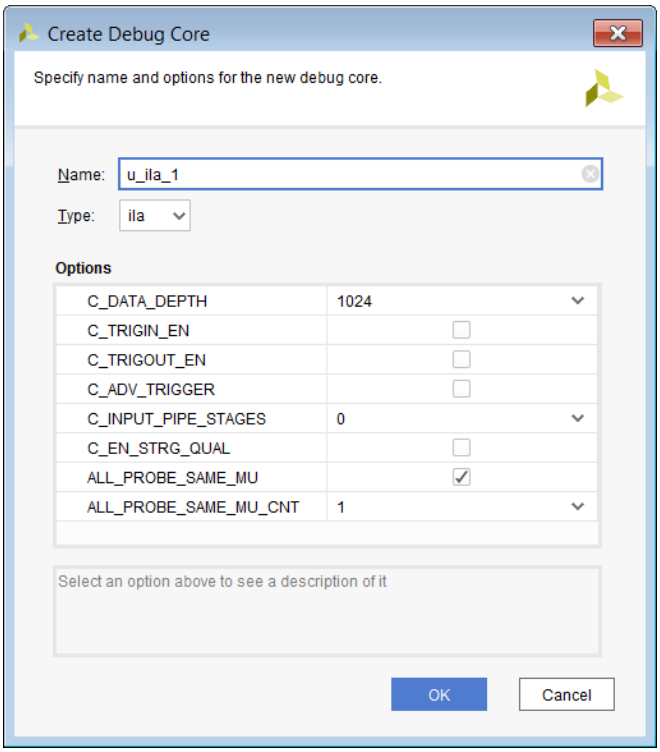
相关信息

在调试核上修改属性

添加、移除和自定义调试核端口

除了添加和移除调试核之外，您也可以添加、移除和自定义每个调试核的端口以满足自己的调试需求。要添加新的调试端口，请执行以下操作：

1. 在“Debug”（调试）窗口中选择调试核。
2. 单击“Create Debug Port”（创建调试端口）打开对话框。
3. 选择或输入端口宽度。
4. 单击“OK”（确定）。
5. 要移除调试端口，请首先在“Debug”（调试）窗口中选中核上的端口，然后选择“Delete”（删除）。



在信号线与调试核之间建立连接和断开连接

您可在“Schematic”（板级原理图）窗口或“Netlist”（网表）窗口上选中信号线和总线（也称为总线信号线）并将其拖放到调试核端口上。这样即可根据需要扩展调试端口以适应所选信号线。您还可右键单击任意信号线或总线，然后单击“Assign to Debug Port”（分配给调试端口）。

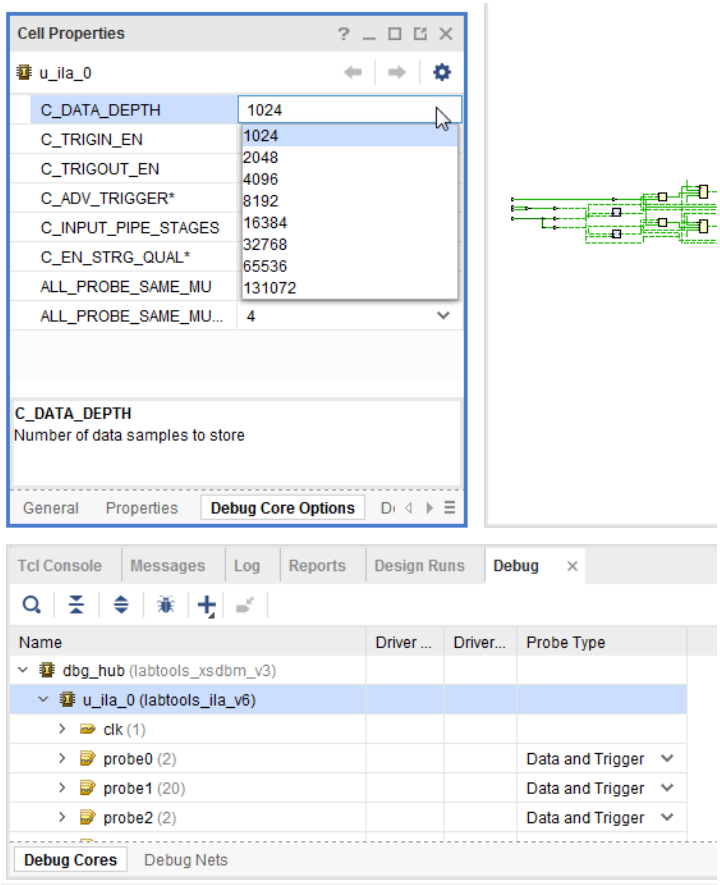
要断开信号线与调试核端口之间的连接，请选中已连接到调试核端口的信号线，然后单击“Disconnect Net”（断开信号线）。

在调试核上修改属性

每个调试核都具有可供更改的属性，您可更改这些属性以自定义其行为。要了解如何在 `debug_core_hub` 调试核上更改属性，请参阅“更改 Debug Hub 核的 BSCAN 用户扫描链”。

您还可在 ILA 调试核上更改属性。例如，要更改 ILA 调试核捕获的样本数量，请执行以下操作：

1. 在“Debug”（调试）窗口中，选中目标 ILA 核（例如，u_ila_0）。
2. 在“Cell Properties”（单元属性）窗口中，选中“Debug Core Options”（调试核选项）视图。
3. 使用 C_DATA_DEPTH 下拉列表，选中期望捕获的样本数量。



在下表中可找到所有 ILA 核属性的完整描述。

表 6：**ILA 调试核属性**

调试核属性	描述	可能的值
C_DATA_DEPTH	ILA 核可存储的数据样本的最大数量。如果增大该值，则会在 ILA 核中占用更多块 RAM，从而对设计性能产生不利影响。	1024（默认值） 2048 4096 8192 16384 32768 65536 131072
C_TRIGIN_EN	启用 ILA 核的 TRIG_IN 端口和 TRIG_IN_ACK 端口。 **注释：**您必须使用高级网表更改命令才能将这些端口连接到设计中的信号线。如果要使用 ILA 触发器输入或输出信号，请考虑使用将 ILA 核添加到设计中的 HDL 例化方法。	false（默认值） true
C_TRIGOUT_EN	启用 ILA 核的 TRIG_OUT 和 TRIG_OUT_ACK 端口。 **注释：**您必须使用高级网表更改命令才能将这些端口连接到设计中的信号线。如果要使用 ILA 触发器输入或输出信号，请考虑使用将 ILA 核添加到设计中的 HDL 例化方法。	false（默认值） true

表 6：**ILA 调试核属性** (续)

调试核属性	描述	可能的值
C_ADV_TRIGGER	启用 ILA 核的高级触发器模式。请参阅“在硬件中调试逻辑设计”以获取有关此功能的更多详细信息。	false（默认值） true
C_MEMORY_TYPE（仅限 Versal）	选择要用于 AXIS-ILA 走线存储器的存储器原语（BRAM 或 UltraRAM）。UltraRAM 适用于块 RAM 使用率较高的设计。	0 (BRAM) 1 (URAM)
C_INPUT_PIPE_STAGES	在 ILA 核的 PROBE 输入上启用额外的管道阶段层级（例如，触发器寄存器）。您可使用此功能来支持 Vivado 工具将 ILA 核布局在远离设计关键部分的位置，从而提升设计的时序性能。	0（默认值） 1 2 3 4 5 6
C_EN_STRG_QUAL	启用 ILA 核的基本捕获控制模式。请参阅“在硬件中调试逻辑设计”以获取有关此功能的更多详细信息。	false（默认值） true
C_ALL_PROBE_SAME_MU	启用 ILA 核的所有 PROBE 输入，使其包含相同数量的比较器（也称为“匹配单元”）。该属性应始终设置为 true。	true（默认值） false（不推荐）
C_ALL_PROBE_SAME_MU_CNT	ILA 核的每个 PROBE 输入的比较器（或匹配单元）数量。所需的比较器数量取决于 C_ADV_TRIGGER 和 C_EN_STRG_QUAL 属性的设置： 如果 C_ADV_TRIGGER 为 false 且 C_EN_STRG_QUAL 为 false，则此项可设置为 1 到 16。 如果 C_ADV_TRIGGER 为 false 且 C_EN_STRG_QUAL 为 true，则此项可设置为 2 到 16。 如果 C_ADV_TRIGGER 为 true 且 C_EN_STRG_QUAL 为 false，则此项可设置为 1 到 16。 如果 C_ADV_TRIGGER 为 true 且 C_EN_STRG_QUAL 为 true，则此项可设置为 2 到 16。 重要：如不遵循上述规则，则可能会在实现期间生成 ILA 核时遇到错误。	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

相关信息

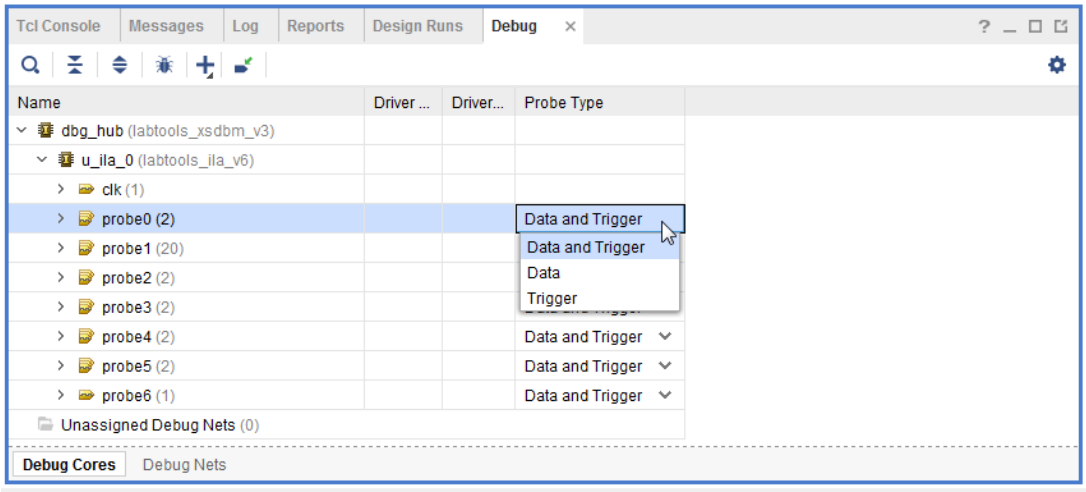
更改 Debug Hub 核的 BSCAN 用户扫描链
在硬件中调试逻辑设计

探针用作为数据和/或触发器

您可在 Vivado 硬件管理器中自定义探针，将其用作为数据和/或触发器。如果探针参与比较值的触发或捕获，则应将其配置为仅限“触发器”探针。这样即可优化 ILA 核使用 BRAM 的方式。通常，如需捕获探针数据，则应将其配置为仅限“数据”探针。如果探针参与比较值的触发并且需要捕获探针数据，则应将其配置为“触发器和数据”探针。

您可使用“Set up Debug”Wizard（设置调试向导）来将探针配置为数据和/或触发器，如下图所示。

图 79：**使用“Set Up Debug”Wizard 将探针配置为数据和/或触发器**

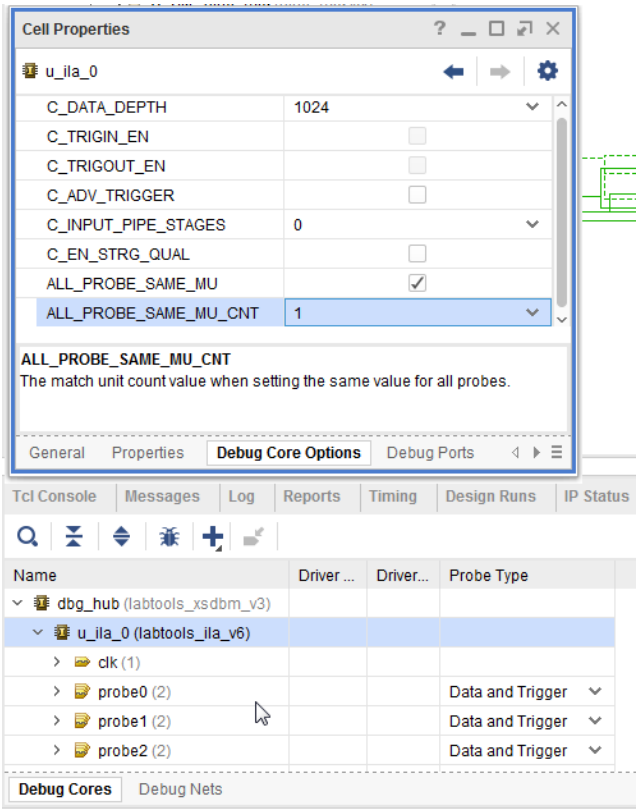


对于包含配置为仅限“数据”的探针的设计，如果您在运行时对其执行器件烧录，则无法使用这些探针来配置触发器或捕获设置条件。反之，在“Waveform”（波形）窗口中，将无法使用配置为仅限“触发器”的探针。

配置所用比较器的数量

在综合后网表上插入 ILA 核后，即可设置用于任意位置的比较器的数量（范围为 1 到 16）。要在 Vivado IDE 中设置比较器数量，请转至 ILA 核的“Debug Core Options”（调试核选项）选项卡，并将 `ALL_PROBE_SAME_MU_CNT` 属性设置为所期望的比较器数量。

图 80：**调试核选项**



或者，也可以在 Tcl 控制台中设置 ALL_PROBE_SAME_MU_CNT 属性，如下所示：

```
set_property ALL_PROBE_SAME_MU_CNT 10 [get_debug_cores u_ila_0]
```

提示：如果启用“Capture Control”（捕获控制），则可选比较器数量范围是 1 到 15 个。有 1 个比较器供捕获控制数据筛选机制使用。

重要提示！在 ILA 中无法使用插入流程来为不同探针设置不同数量的比较器。AMD 建议您使用 HDL 例化流程来为不同探针设置不同数量的比较器。

使用 XDC 命令来插入调试核

除了使用“Set up Debug”Wizard（设置调试向导）外，您也可以使用 XDC 命令来创建、连接调试核并将其插入已综合的设计网表。在 Tcl 控制台中输入 XDC 命令，并遵循以下步骤进行操作：

1. 打开已综合的设计网表，此网表来自名为 synth_1 的综合运行。

```
open_run synth_1
```

重要提示！仅当打开已综合的设计网表后，以下步骤中的 XDC 命令才有效。

2. 创建 ILA 核的黑盒。

```
create_debug_core u_ila_0 ila
```

3. 设置 ILA 核的各属性。

```
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0] set_property
C_TRIGIN_EN false [get_debug_cores u_ila_0] set_property C_TRIGOUT_EN
false [get_debug_cores u_ila_0] set_property C_ADV_TRIGGER false
[get_debug_cores u_ila_0] set_property C_INPUT_PIPE_STAGES 0
[get_debug_cores u_ila_0] set_property C_EN_STRG_QUAL false
[get_debug_cores u_ila_0] set_property ALL_PROBE_SAME_MU true
[get_debug_cores u_ila_0] set_property ALL_PROBE_SAME_MU_CNT 1
[get_debug_cores u_ila_0]
```

4. 将 ILA 核的 clk 端口宽度设为 1，并将其连接到目标时钟信号线。

```
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
```

注释：您无需创建 ILA 核的 clk 端口，因为此端口将由 `create_debug_core` 命令自动创建。

重要提示！调试核的所有调试端口名称均为小写。使用大写或混合大小写的调试端口名称将导致出错。

5. 将 probe0 端口的宽度设置为您计划连接到此端口的信号线的数量。

注释：您无需创建 ILA 核的首个探测端口 (probe0)，因为此端口将由 `create_debug_core` 命令自动创建：set_property port_width 1 [get_debug_ports u_ila_0/probe0]

6. 将 probe0 端口连接到您想要连接到该端口的信号线。

```
connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
```

7. （可选）创建更多探测端口、设置其宽度并将其连接到您要调试的信号线。

```
create_debug_port u_ila_0 probe set_property port_width 2
[get_debug_ports u_ila_0/probe1] connect_debug_port u_ila_0/probe1
[get_nets [list {A[0]} {A[1]}]]
```

如需获取有关这些 Tcl 命令以及有关其他相关 Tcl 命令的更多信息，请在 Vivado Design Suite 的 Tcl 控制台中输入 `help -category ChipScope`。

运行调试 XDC 命令后保存约束

使用“Set up Debug”Wizard（设置调试向导）、使用 Vivado Design Suite 创建调试核或调试端口和/或运行以下 XDC 命令后，需保存约束：

- `create_debug_core`
- `create_debug_port`
- `connect_debug_port`
- `set_property`（针对任意 `debug_core` 或 `debug_port` 对象）

对应 XDC 命令将保存到含后缀 `_debug.xdc` 的约束文件，并在实现期间，用于插入和连接调试核。

重要提示！在工程模式下保存约束可能导致综合步骤和实现步骤过期。但您无需重新综合设计，因为仅在实现期间才使用这些调试 XDC 约束。您可通过选中“Design Runs”（设计运行）窗口、右键单击综合运行轮次（例如，`synth_1`），然后选中“Force up-to-date”（强制更新）来强制将综合步骤更新至最新状态。

实现设计

完成插入、连接并自定义调试核后，即可着手实现设计（请参阅“对包含调试核的设计执行实现”）。

相关信息

对包含调试核的设计执行实现

在非工程模式下执行调试核插入

调试核可在工程模式或非工程模式下插入。以下 Tcl 脚本样本演示了如何在所探测的设计中创建调试核、设置调试核属性以及将调试核探针连接到信号。在非工程模式下，调试核插入操作需在设计综合之后但在 opt_design 步骤之前执行，如下所示：

重要提示！仅限 ILA 核才支持调试核插入。

以下 Tcl 脚本提供了在非工程流程中使用调试核插入命令的示例。

```
#read relevant design source files read_vhdl [glob ./*.vhdl] read_verilog
[ glob ./Sources/*.v ] read_xdc ./target.xdc #Synthesize Design
synth_design -top top -part xc7k325tffg900-2 #Create the debug core
create_debug_core u_ila_0 ila #set debug core properties set_property
C_DATA_DEPTH 1024 [get_debug_cores u_ila_0] set_property C_TRIGIN_EN false
[get_debug_cores u_ila_0] set_property C_TRIGOUT_EN false [get_debug_cores
u_ila_0] set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0] set_property
C_EN_STRG_QUAL false [get_debug_cores u_ila_0] set_property
ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0] set_property
ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0] #connect the probe ports
in the debug core to the signals being probed in the design set_property
port_width 1 [get_debug_ports u_ila_0/clk] connect_debug_port u_ila_0/clk
[get_nets [list clk ]] set_property port_width 1 [get_debug_ports u_ila_0/
probe0] connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
create_debug_port u_ila_0 probe #Optionally, create more probe ports, set
their width, #  and connect them to the nets you want to debug #Implement
design opt_design place_design report_drc -file ./placed_drc_rpt.txt
report_timing_summary -file ./placed_timing_rpt.txt route_design report_drc
-file ./routed_drc_rpt.txt report_timing_summary -file ./
routed_timing_rpt.txt write_bitstream
```

HDL 例化调试探测流程概述

HDL 例化探测流程涉及在 HDL 设计源代码中直接手动自定义、例化和连接各种调试核组件。下表中显示了 Vivado 工具中此流程所支持的新调试核。

表 7：**Vivado IP 目录中可用于 HDL 例化探测流程的调试核**

调试核	版本	描述	运行时分析器工具
ILA (Integrated Logic Analyzer)	v6.2	此调试核用于触发硬件事件并以系统级速度捕获数据。	Vivado Logic Analyzer

表 7：**Vivado IP 目录中可用于 HDL 例化探测流程的调试核** (续)

调试核	版本	描述	运行时分析器工具
VIO (Virtual Input/ Output)	v3.0	此调试核用于按 JTAG 链扫描速率监控或控制设计中的信号。	Vivado Logic Analyzer
JTAG-to-AXI Master	v1.2	此调试核用于生成 AXI 传输事务，这些传输事务用于与硬件内运行的系统中的各种 AXI4 和 AXI4-Lite 从核进行交互。	Vivado Logic Analyzer

新的 ILA 核相比于传统 ILA v1.x 核具有以下 2 大优势：

- 可搭配集成的 Vivado Logic Analyzer 功能一起使用（请参阅“在硬件中调试逻辑设计”）。
- 无需 ICON 核实例或连接。

相关信息

在硬件中调试逻辑设计

使用 HDL 例化调试探测流程

执行 HDL 例化流程所需的步骤如下：

1. 自定义并生成 ILA 和/或 VIO 调试核，其中包含正确数量的探测端口，与要探测的信号对应。
2. （可选）自定义并生成 JTAG-to-AXI Master 调试核，并将其连接到设计中的 AXI 外设或互连核的 AXI 从接口。
3. 对包含调试核的设计执行综合。
4. （可选）修改 Debug Hub 核属性。
5. 对包含调试核的设计执行实现。

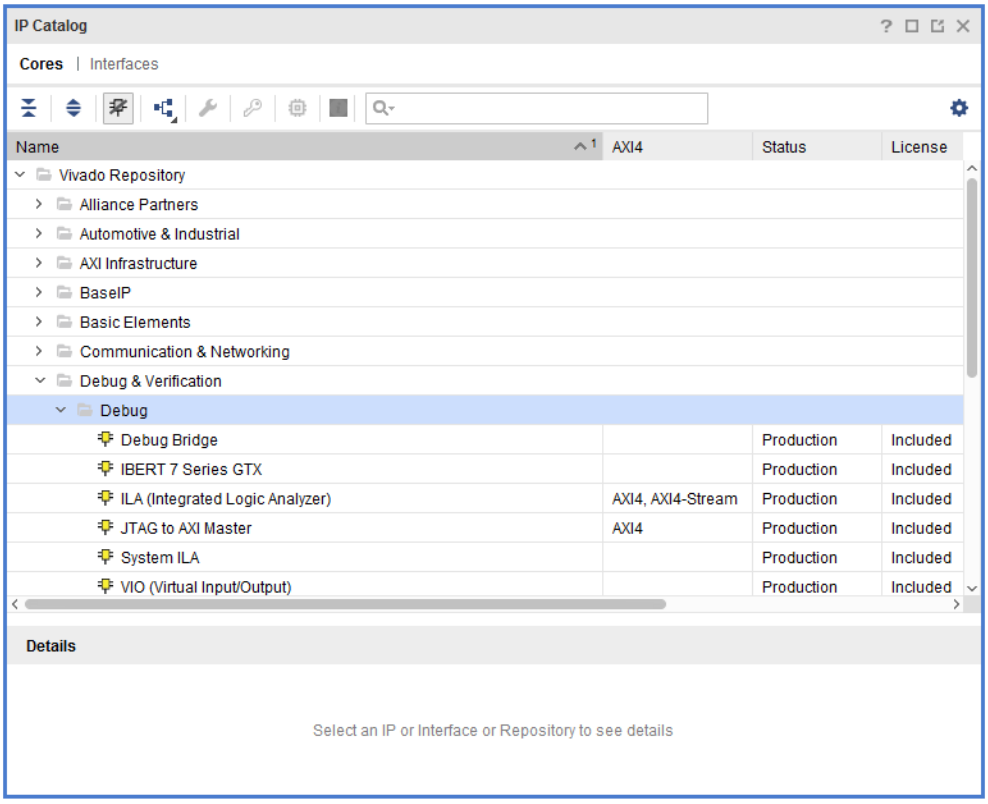
自定义和生成调试核

“Project Manager”（工程管理器）中的“IP Catalog”（IP 目录）按钮可用于查找、选择和自定义所需调试核。调试核位于 IP 目录的“Debug & Verification > Debug”（调试和验证 > 调试）类别下，请参阅下图。您可双击 IP 核或者右键单击“Customize IP”（自定义 IP）菜单选项来自定义调试核。

- 如需了解有关自定义 ILA 核的更多信息，请参阅《Integrated Logic Analyzer LogiCORE IP 产品指南》(PG172)。
- 如需了解有关自定义 VIO 核的更多信息，请参阅《Virtual Input/Output LogiCORE IP 产品指南》(PG159)。
- 如需了解有关自定义 JTAG-to-AXI Master 核的更多信息，请参阅《JTAG to AXI Master LogiCORE IP 产品指南》(PG174)。

自定义核之后，请单击“IP customization”Wizard（IP 自定义向导）中的“Generate”（生成）按钮。这样即可生成已自定义的调试核，并将其添加到工程的“Sources”（源）视图中。

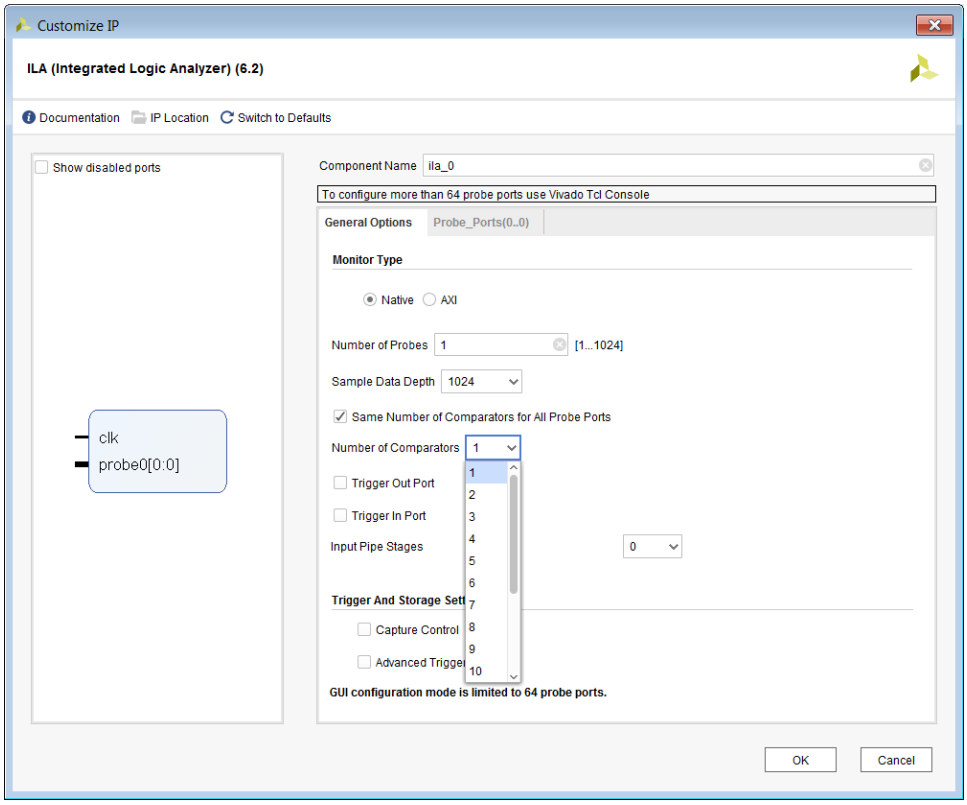
图 81：**IP 目录中的调试核**



配置所用比较器的数量

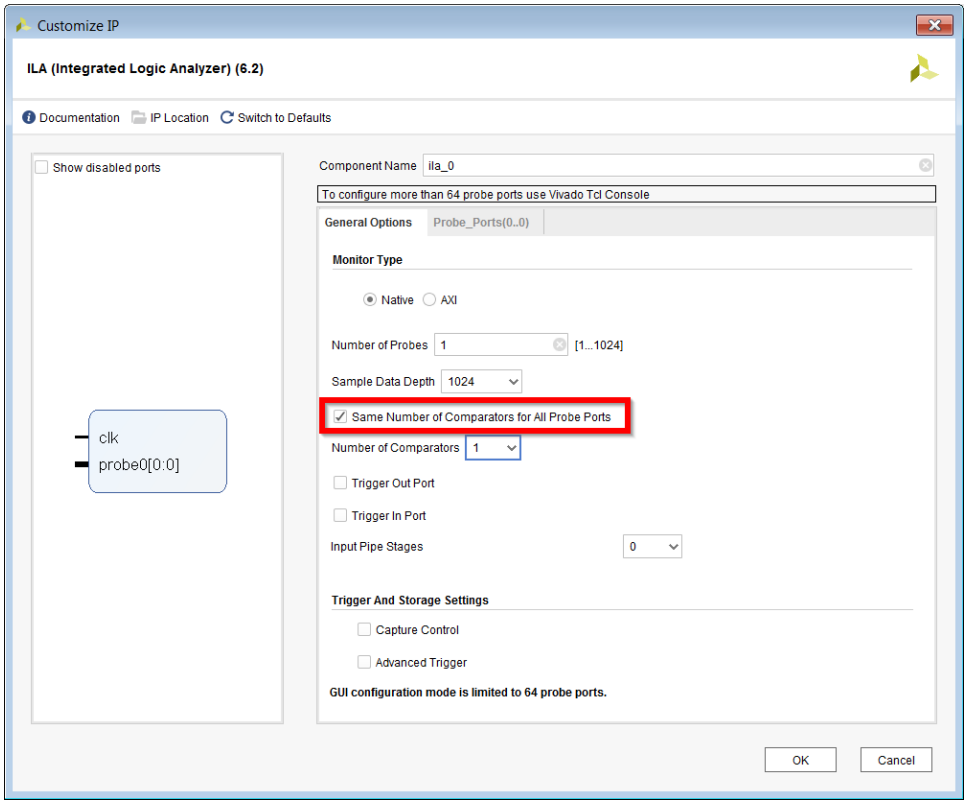
在自定义 ILA IP 的进程期间，您可设置所用的比较器的数量。允许的范围为 1 至 16。可为 ILA IP 中的所有探针设置公用比较器的数量。

图 82：**常规选项中的 ILA IP 比较器**



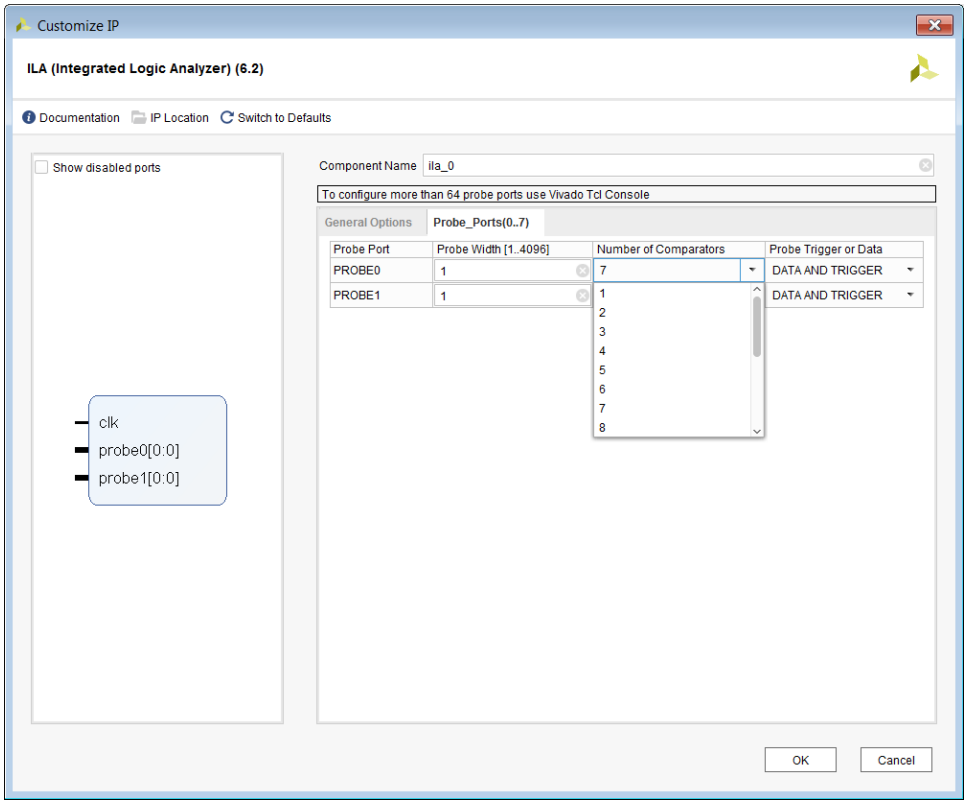
也可以为每个 IP 设置比较器，如下所示。任一 ILA 内均可包含多个不同宽度的探针。为此，您需要取消勾选“General Options”（常规选项）下的“Same Number of Comparators for All Probe Ports”（所有探针端口采用相同数量的比较器）字段。

图 83： **“Same Number of Comparators for All Ports”字段**



您可通过如下方式来为每个探针设置要使用的比较器的准确数量：选中“Probe_Ports”选项卡并在“Number of Comparators”（比较器数量）字段中设置所期望的比较器数量。

图 84：**Number of Comparators**



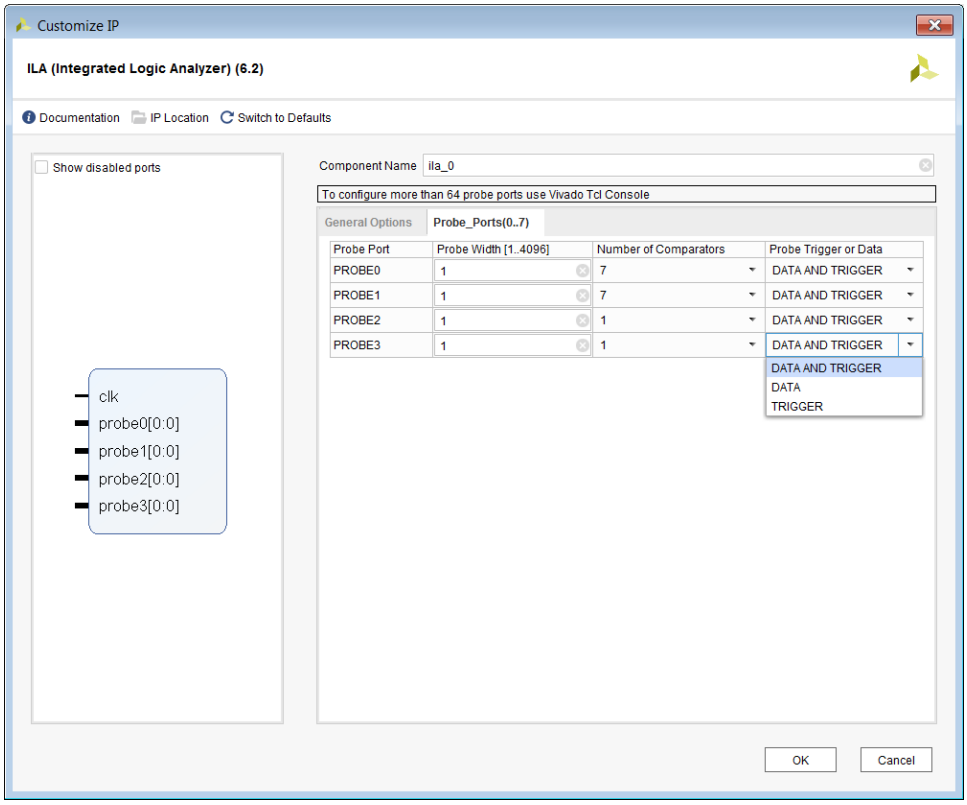
提示：如果启用“Capture Control”（捕获控制），则可选比较器数量范围是 1 到 15 个。有 1 个比较器供捕获控制数据筛选机制使用。

提示：根据所选比较器数量，该工具会自动重新计算您可在 ILA IP 中使用的探针的数量。每个 ILA 允许的比较器最大数量为 1024。

探针作为数据或触发器

在“ILA IP Configuration”Wizard（ILA IP 配置向导）中，可将探针配置为数据和/或触发器，如下图所示。

图 85：**将探针配置为触发器和数据**



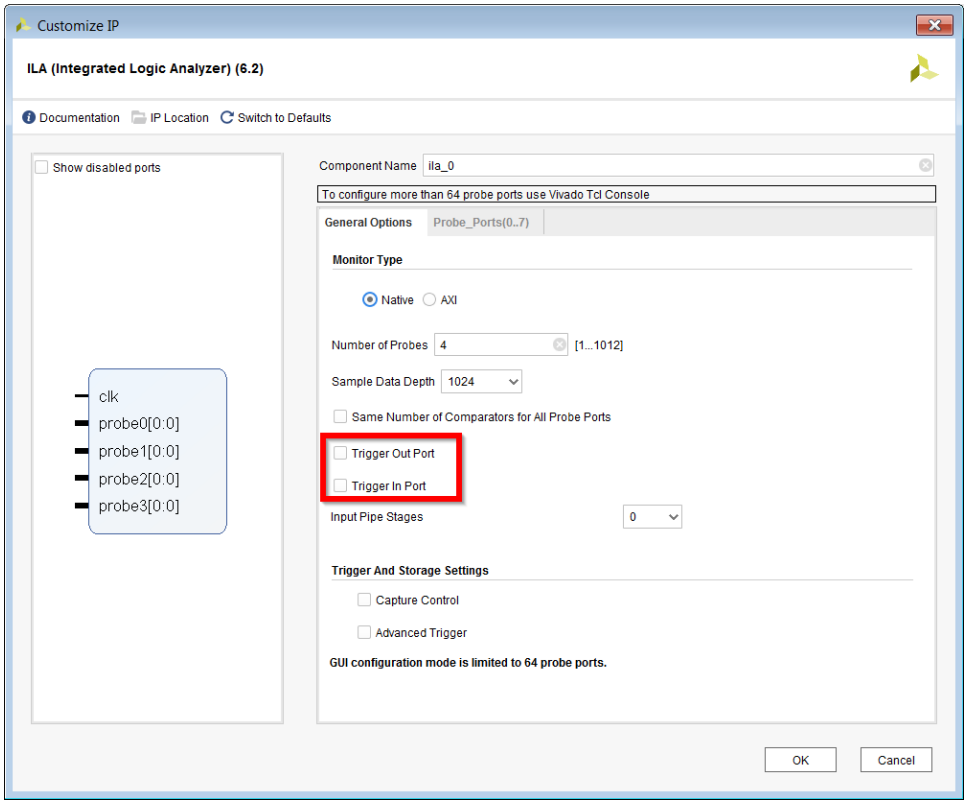
如果探针参与比较值的触发或捕获，则应将其配置为仅限“触发器”探针。这样即可优化 ILA 核使用 BRAM 的方式。通常，如需捕获探针数据，则应将其配置为仅限“数据”探针。如果探针参与比较值的触发并且需要捕获探针数据，则应将其配置为“触发器和数据”探针。

ILA 交叉触发

ILA 交叉触发功能支持在 ILA 核之间以及在 ILA 核与处理器（如 AMD Zynq™ 7000 SoC）之间进行交叉触发。如果要在位于不同时钟域中的 2 个 ILA 核之间执行触发，或者要在处理器与 ILA 核之间执行硬件/软件交叉触发，那么可使用该功能。

要使用交叉触发功能，在核生成时，应配置 ILA 核心以使其包含专用触发器输入端口（TRIG_IN 和 TRIG_IN_ACK）和专用触发器输出端口（TRIG_OUT 和 TRIG_OUT_ACK）。如果要使用 ILA 触发器输入或输出信号，则必须使用将 ILA 核添加到设计中的 HDL 例化方法。

图 86：**ILA 交叉触发功能**



TRIG_OUT_ACK 是 ILA 核（另一个 ILA、用户设计或处理器）的指示信号，表明已正确接收到 TRIG_OUT，它会导致 ILA 在接收到 TRIG_OUT_ACK 时拉低 TRIG_OUT 信号。

换言之，TRIG_OUT 会保持高电平直至 TRIG_OUT_ACK 可用为止。如果 TRIG_OUT_ACK 信号绑定到低电平，那么 TRIG_OUT 会保持高电平，直至用户重新装备 ILA 为止。只有 TRIG_OUT 会转至低电平。如果 TRIG_OUT_ACK 绑定到低电平，那么您可重新装备 ILA。

下图显示了典型的交叉触发设置，其中 ILA2 交叉触发到 ILA1 内。ILA2 的 TRIG_OUT 信号连接到 ILA1 的 TRIG_IN 信号。ILA1 的 TRIG_IN_ACK 信号则连接到 ILA2 的 TRIG_OUT_ACK 信号。

```
(ILA 2) trig_out -> (ILA 1) trig_in (ILA 1) trig_in_ack -> (ILA 2)
trig_out_ack
```

图 87：**典型交叉触发设置**

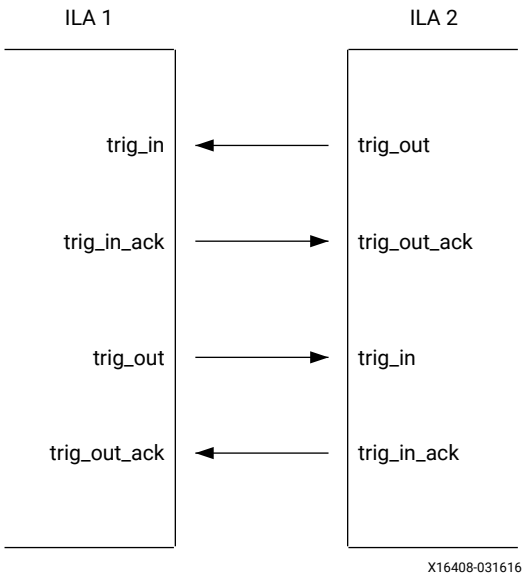
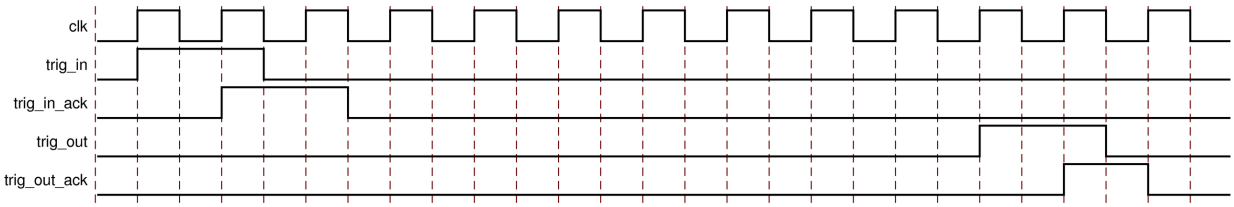


图 88：**ILA 交叉触发时序**



- 假定驱动 `trig_in` 端口的逻辑与 ILA clk 保持同步。
- 当 `trig_in` 断言有效后，`trig_in_ack` 信号需经过 1 个时钟周期后才能断言有效。
- 当使用 `trig_in` 或者满足触发条件后，`trig_out` 信号需经过 9 个时钟周期才能断言有效。
- 仅当触发器信号断言无效后，`trig_in_ack` 和 `trig_out_ack` 信号才会转至低电平。

如需获取有关在 FPGA 互连结构与 Zynq 7000 SoC 处理器之间使用交叉触发功能的详细教程，请参阅《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940)。

例化调试核

生成调试核后，在 HDL 源代码中将其例化，然后将其连接到要探测的信号以便对其进行调试。以下是 Verilog HDL 源文件中的 ILA 实例的示例：

```
u_ila_0 (   .clk(clk),   .probe0(counterA),   .probe1(counterB),
 .probe2(counterC),   .probe3(counterD),   .probe4(A_or_B),   .probe5(B_or_C),
 .probe6(C_or_D),   .probe7(D_or_A) );
```

注释：不同于旧的 VIO 和 ILA v1.x 核，新的 ILA 核实例无需连接至 ICON 核实例。而改为将 Debug Hub 核 (`dbg_hub`) 自动插入已综合的设计网表，以便在新的 ILA 核与 JTAG 扫描链之间提供连接。

对包含调试核的设计执行综合

下一步是在 Vivado Design Suite 中单击“Run Synthesis”（运行综合）或者运行以下 Tcl 命令来对包含调试核的设计执行综合操作：

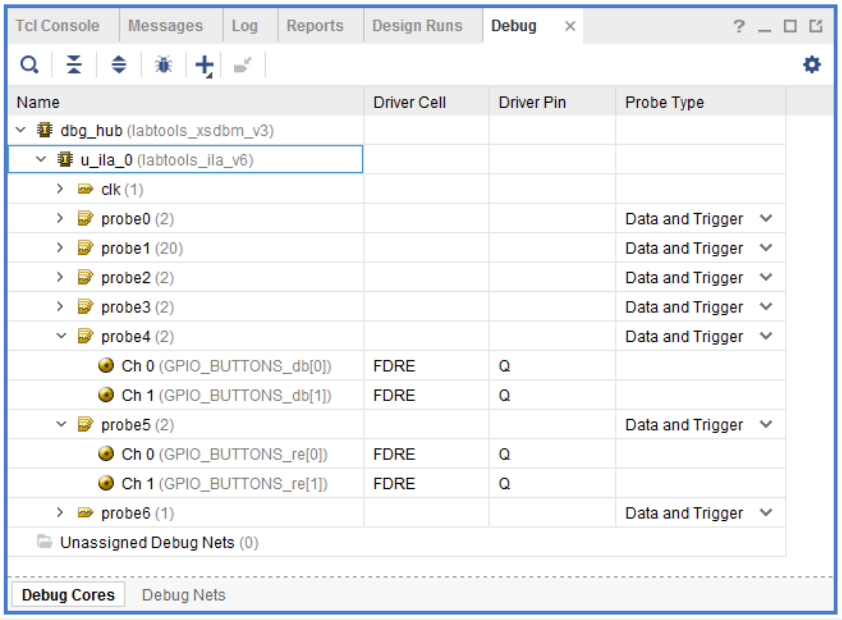
```
launch_runs synth_1 wait_on_run synth_1
```

您也可以使用 synth_design Tcl 命令来对设计执行综合。如需了解有关各种设计综合方法的更多详细信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

在已综合的设计内查看调试核

完成设计综合后，您可打开已综合的设计，以查看调试核并修改其属性。单击“Flow Navigator”中的“Open Synthesized Design”（打开已综合的设计）以打开已综合的设计，然后选择“Debug”（调试）窗口布局以查看“Debug”（调试）窗口，其中显示已连接到 Debug Hub 核 (dbg_hub) 的 ILA 调试核（如下图所示）。

图 89：**显示 ILA 核以及 Debug Hub 核的“Debug”窗口**



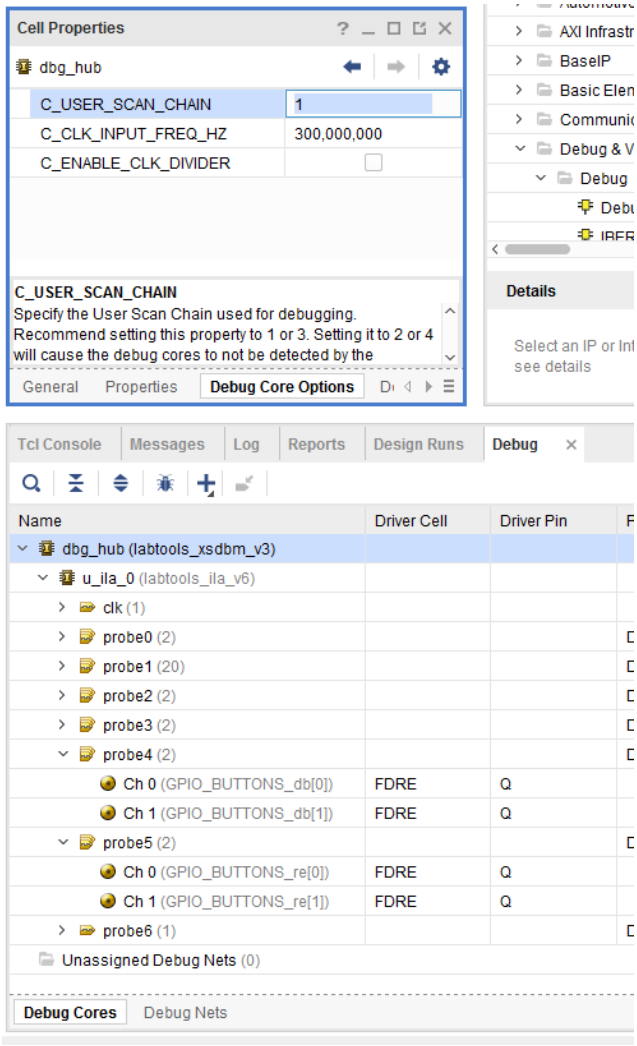
更改 Debug Hub 核的 BSCAN 用户扫描链

您可通过如下方式来查看和更改 Debug Hub 核的 BSCAN 用户扫描链索引：选中“Debug”（调试）窗口中的“dbg_hub”，选中“Properties”（属性）窗口中的“Debug Core Options”（调试核选项）视图，然后更改“C_USER_SCAN_CHAIN”属性的值（请参阅下图）。

重要提示！对于 Debug Hub 核，C_USER_SCAN_CHAIN 默认值为 1。如果对于 Debug Hub 核使用的扫描链值不为 1，则必须在器件上通过硬件管理器来手动更改这些值。请参阅“硬件器件烧录”以获取更多详细信息。

重要提示！如果您计划使用 MicroBlaze 或 MicroBlaze V Debug Module Debug Module (MDM)，或者其他将 BSCAN 原语与 Vivado 逻辑调试核搭配使用的 IP，那么就需要将 dbg_hub 的 C_USER_SCAN_CHAIN 属性设为与其他 IP 边界扫描链设置不冲突的用户扫描链。否则，可能导致后续实现流程中出错。

图 90：**更改 Debug Hub 核的用户扫描链属性**



相关信息

硬件器件烧录

IP integrator 中的调试流程

Vivado IP integrator 中的 System ILA IP 允许您对 FPGA 或自适应 SoC 上的实现后设计执行系统内调试。如需监控 IP integrator 块设计中的接口和信号，请使用此功能。此功能支持您在 Vivado 硬件管理器中对 AXI 读写、数据和地址通道事件以及 AXI 读写传输事务进行调试。

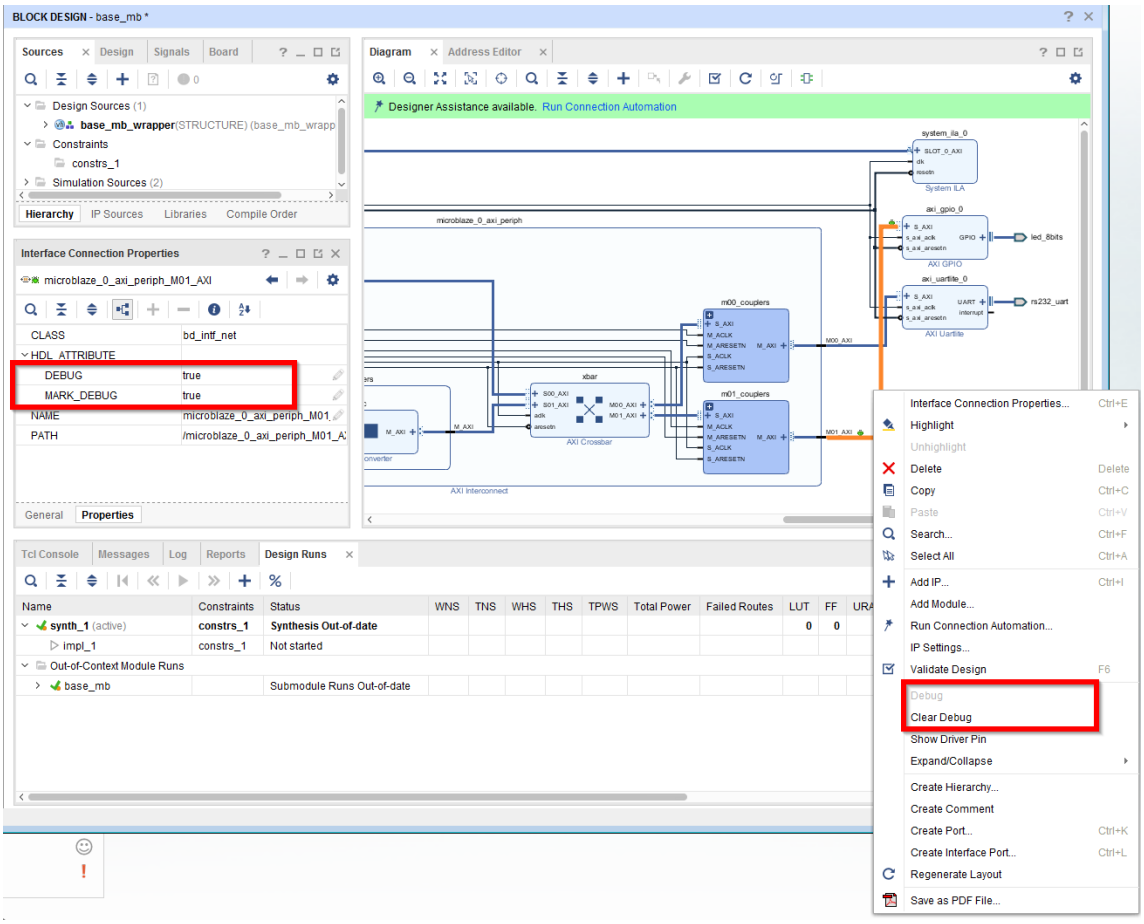
注释：在 Versal 器件上，使用 ILA 核时支持所有 System ILA 功能，并且可通过将“ILA Input Type”（ILA 输入类型）从 `Net Probes` 更改为 `Interface Monitor` 来选择相应的功能。

请访问此链接并参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994) 中的相应内容，以了解在块设计中调试接口和/或信号线的步骤。

在 IP integrator 块设计中调试信号线和接口

在 IP integrator 块设计画布中，可调试信号线和接口。您可在块设计中右键单击接口或信号线，然后单击“Debug”（调试），如下所示。这样即可将“Debug”和“MARK_DEBUG”属性设为 true。此外，这样还可启用“Designer Assistance”（设计辅助）以运行“Connection Automation”（自动连接功能），以供您选择连接到 System Interface ILA 核的信号线和/或接口，并可自定义调试核的各属性。

图 91：**IP integrator 标记调试**

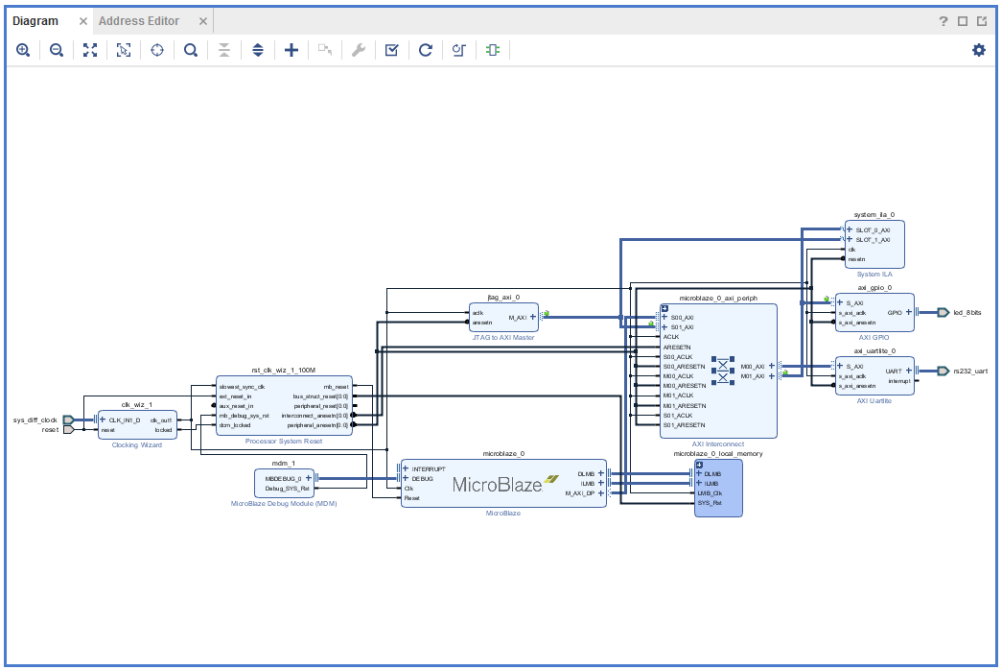


要清除调试属性的信号线和/或接口，请右键单击该信号线或接口，然后单击“Clear Debug”（清除调试）。

在已综合的设计内查看 System ILA 调试核

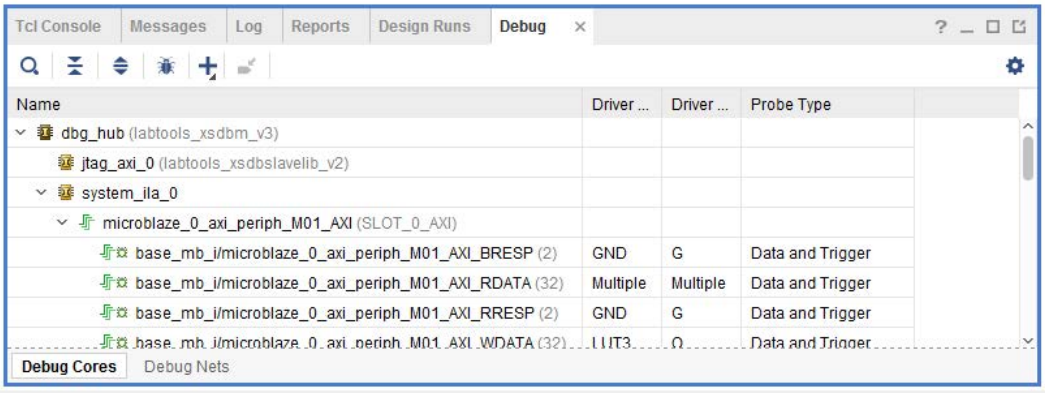
IP integrator 块设计中的 System ILA IP 必须例化。下图所示快照中的块设计包含 2 个已例化的调试核：System ILA 和 JTAG to AXI Master IP 核。

图 92：**块设计**



在此块设计完成确认和综合后，您可在已综合的设计中打开“Debug”（调试）窗口以查看已例化并插入设计的调试核。System ILA 和 JTAG to AXI Master 调试核如下所示：

图 93：**System ILA 和 JTAG to AXI Master 调试核**



如需了解有关如何使用这些接口在硬件管理器中进行调试以及如何利用 AXI 事件级调试的详细信息，请参阅“在硬件管理器中调试 AXI 接口”。

相关信息

在硬件管理器中调试 AXI 接口

对包含调试核的设计执行实现

Vivado 软件最初以黑盒形式创建 Debug Hub 核。必须先实现该核，然后才能运行布局器和布线器。

实现设计

在 Vivado Design Suite 中单击“Run Implementation”或者运行以下 Tcl 命令来实现包含调试核的设计：

```
launch_runs impl_1 wait_on_run impl_1
```

您也可以使用实现命令 `opt_design`、`place_design` 和 `route_design` 来实现设计。如需了解有关各种设计实现方法的更多详细信息，请参阅《Vivado Design Suite 用户指南：实现》(UG904)。

ILA 核与时序注意事项

ILA 核的配置会对能否满足整体设计时序目标产生影响。请根据建议进行操作，尽可能减少对时序的影响：

- 请审慎选择探针宽度。随探针宽度增加，对资源使用率和时序的影响也会增大。
- 请审慎选择 ILA 核数据深度。随数据深度增加，对块 RAM 资源使用率和时序的影响也会增大。
- 请确保为 ILA 核选择的时钟均为自由运行的时钟。否则可能造成在器件上加载设计时无法与调试核通信。
- 在添加调试核之前完成设计上的时序收敛。AMD 不建议使用调试核来调试时序相关问题。
- 如果添加调试核之后，设计不满足时序要求，并且时序失败发生在 ILA 或 AXIS-ILA 核中，那么请尝试增大输入流水线阶段数 (C_INPUT_PIPE_STAGES)。
- 如果添加调试核之后，设计不满足时序要求，并且时序失败发生在 AXIS-ILA 核中，那么请尝试将存储目标更改为 UltraRAM (URAM)，因为这样可以为块 RAM (BRAM) 控制信号放宽时序要求。
- 如果添加调试核之后，设计不满足时序要求，并且在 ILA 或 AXIS-ILA 核中发生时序失败，那么请尝试采用不同的实现策略，例如，`Performance_Explore` 或 `Performance_ExtraTimingOp`。
- 请确保输入到 ILA 核的时钟与正在探测的信号同步。否则在设计烧录到器件中时会产生时序问题并导致通信失败。
- 对于 Versal 架构，如果在添加调试核之后观测到时序失败，请尝试对连接到 AXI4-Debug Hub 的时钟使用介于 100 MHz 到 250 MHz 之间的时钟频率，因为这样即可在连接到此 AXI4-Debug Hub 的所有调试核上放宽 AXI4-Stream 连接的时序要求。
- 在硬件上运行设计之前请确保设计已满足时序要求。否则会导致结果不可靠。
- 对于非 Versal 架构，请确保连接到 `dbg_hub` 的时钟为自由运行的时钟。否则可能造成在器件上加载设计时无法与调试核通信。可使用 `connect_debug_port` Tcl 命令将 Debug Hub 的 `clk` 管脚连接到自由运行的时钟。
- 对于非 Versal 架构，如果仍发现因添加 ILA 调试核而导致的时序劣化，并且关键路径位于 `dbg_hub` 中，请执行以下步骤：
  1. 打开已综合的设计。
  2. 找到网表中的 `dbg_hub` 单元。
  3. 转至 `dbg_hub` 的“Properties”（属性）选项卡。
  4. 找到 `C_CLK_INPUT_FREQ_HZ` 属性。

5. 将其设置为连接到 dbg_hub 的时钟频率 (Hz)。
6. 找到 C_ENABLE_CLK_DIVIDER 属性并将其启用。
7. 重新执行设计实现。

调试核时钟设置指南

注释：以下章节适用于 7 系列、UltraScale 和 UltraScale+ 器件。Versal 调试核使用基于 AXI 的连接，且不受本章中的时钟设置准则的约束。

Vivado 硬件管理器使用 JTAG 接口来与 Vivado Debug Hub 核进行通信，后者可在 FPGA 的 JTAG 边界扫描 (BSCAN) 接口与 Vivado 调试核之间提供接口。

- “JTAG Clock”（JTAG 时钟）：此时钟可用于同步 JTAG 边界扫描 (BSCAN) 接口的内部状态机操作。连接到目标器件时，可在 Vivado 硬件管理器中选择 JTAG 时钟频率。如果您的设计包含调试核，请确保 Debug Hub 时钟频率至少是 JTAG 时钟频率的 2.5 倍。

  您可使用“Open New Hardware Target”Wizard（打开新硬件目标向导）或者使用以下 Tcl 命令来修改 JTAG 频率：

  ```
  set_property PARAM.FREQUENCY 250000 [get_hw_targets */xilinx_tcf/Digilent/210203327962A]
  ```

- “Debug Hub Clock”（Debug Hub 时钟）：

  Vivado Debug Hub 核，可在 FPGA 的 JTAG 边界扫描 (BSCAN) 接口与 Vivado 调试核之间提供接口。在设计实现步骤中，如果在设计中检测到调试核，那么 Vivado IDE 会自动插入 Debug Hub 核。Vivado IDE 会在设计实现步骤中选择驱动 Debug Hub 核的时钟。

  AMD 建议将 Debug Hub 时钟频率设置为约 100 MHz 或更低，因为 JTAG 时钟速度不需要特别高的频率。

  您可使用以下 Tcl 命令来更改 Debug Hub 时钟。

  ```
  connect_debug_port dbg_hub/clk [get_nets <clock net name>]
  ```

  注释：您需要在设计完成综合后且实现之前运行此命令。

  您还可以使用以下 Tcl 命令将 Debug Hub 时钟频率降低至 100 MHz。

  ```
  set_property C_CLK_INPUT_FREQ_HZ 200000000 [get_debug_cores dbg_hub]
  set_property C_ENABLE_CLK_DIVIDER true [get_debug_cores dbg_hub]
  ```

  注释：您需要在设计完成综合后且实现之前运行此命令。对于时钟速度极高的设计，建议采用此方法。此命令支持在 Debug Hub 核内部包含基于 MMCM 的时钟分频器，以使时钟频率达到 100 MHz。

调试核时钟

Vivado IP 目录中可用的所有调试核都需要时钟，以确保与受监控的输入探针或者由调试核驱动的任意输出信号保持同步。在核发现与调试测量阶段中，时钟应可自由运行并保持稳定。并且时钟应与受监控或受驱动的信号保持同步。否则可能导致一整个周期内的数据不精确。

Debug Hub IP 用于在主机之间（通过支持串行接口的 BSCAN 原语）进行桥接以及在芯片上的调试核之间（通过支持并行接口的 XSDB 接口）进行桥接。BSCAN 原语时钟用于以串行方式将进出芯片的数据切换到 Debug Hub IP。Debug Hub IP 会收集数据，并使用 Debug Hub 时钟将其发送至并行接口上的所有调试核，反之亦然。如有任一调试核时钟未自由运行或者不稳定，则将发生数据损坏，导致出现“`Debug Cores not detected`”消息。为了避免数据损坏，重要的是确保 JTAG 时钟和 Debug Hub 时钟在调试核检测进程中保持稳定并自由运行。

1. Debug Hub 时钟必须自由运行并保持稳定。AMD 建议从已正确约束并且已满足时序的时钟驱动器来驱动时钟。
2. 如果从 MMCM/PLL 驱动时钟，那么在执行任何调试核测量之前，请确保 MMCM/PLL LOCKED 信号处于高电平。如果时钟连接到 Debug Hub 或任一调试核并且在调试操作中间 MMCM/PLL LOCKED 信号转换为 0，那么时钟可能发生显著抖动，从而可能导致调试逻辑出现不可预测的行为。
3. 为了检测调试核，请使用满足上述要求的核与捕获的数据来执行测量。所有关联的时钟都必须自由运行并保持稳定。

下表列出了各调试阶段以及特定阶段内所需的时钟。

表 8：**调试阶段时钟要求**

调试阶段	JTAG 时钟	Debug Hub 时钟	调试核时钟 [2]
连接到目标	稳定 [1]	不适用	不适用
烧录	稳定 [1]	不适用	不适用
调试核发现	稳定 [1]	稳定	不适用
调试核测量 [3]	稳定 [1]	稳定 [1]	稳定

注释：

1. 稳定时钟：事件期间不发生暂停或停止的时钟。
2. 假定调试核时钟不同于 Debug Hub 时钟。
3. 调试核测量阶段包含对调试核上的属性执行 `get` 或 `set` 的所有步骤。

Vivado 硬件管理器时钟设置相关错误消息

如果 JTAG 时钟处于不活动或不可用状态，那么您将无法连接到硬件目标。

如果 Debug Hub 时钟处于不活动或不可用状态，那么 Vivado 硬件管理器会发出以下错误消息：

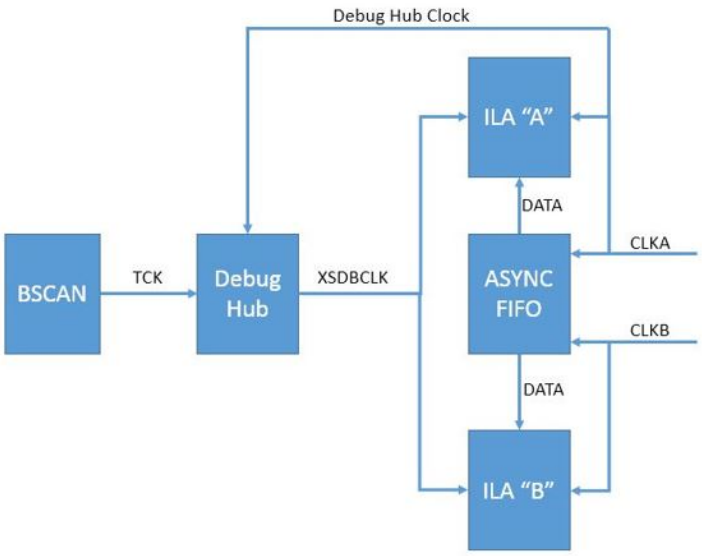
```
INFO: [Labtools 27-1434] Device xxx (JTAG device index = 0) is programmed
with a design that has no supported debug core(s) in it. WARNING: [Labtools
27-3123] The debug hub core was not detected at User Scan Chain 1 or 3.
Resolution: 1. Make sure the clock connected to the debug hub (dbg_hub)
core is a free running clock and is active OR 2. Manually launch hw_server
with -e "set xsdb-user-bscan <C_USER_SCAN_CHAIN scan_chain_number>" to
detect the debug hub at User Scan Chain of 2 or 4. To determine the user
scan chain setting, open the implemented design and use: get_property
C_USER_SCAN_CHAIN [get_debug_cores dbg_hub].
```

如有任何调试核时钟处于不活动或不可用状态，那么 Vivado 硬件管理器会发出以下错误消息：

```
INFO: [Labtools 27-2302] Device xxx (JTAG device index = 1) is programmed
with a design that has 1 ILA core(s). CRITICAL WARNING: [Labtools 27-1433]
Device xxx (JTAG device index = 1) is programmed with a design that has an
unrecognizable debug core (slave type = 17) at user chain = 1, index = 0.
Resolution: 1) Ensure that the clock signal connected to the debug core
and/or debug hub is clean and free-running. 2) Ensure that the clock
connected to the debug core and/or debug hub meets all timing constraints.
3) Ensure that the clock connected to debug core and/or debug hub is faster
than the JTAG clock frequency.
```

下图提供了含 2 个 ILA 核的设计示例：

图 94：**调试核时钟设置示例**



此设计示例包含 2 个 ILA 核：ILA“A”和 ILA“B”。

此调试网络的时钟设置拓扑结构如下所示。当设计烧录到器件中之后，Vivado 硬件管理器会尝试在设计中发现存在的 Debug Hub 核。而 Debug Hub 则会尝试发现并考量连接到自己的所有调试核。在此设计中，调试核为 ILA“A”和 ILA“B”。

注释：CLKA 同时驱动 ILA“A”和 Debug Hub 核。CLKB 则驱动 ILA“B”调试核。

连接到目标并进行器件烧录时，JTAG clk 应处于活动状态。在“调试核发现阶段”，应有自由运行且保持稳定的时钟，用于驱动 Debug Hub 核，在此例中即 CLKA。在“调试核测量阶段”（涉及在调试核上获取/设置属性的任何操作），JTAG、Debug Hub 和调试核时钟应处于活动状态。如果要触发并捕获 ILA“B”上的数据，那么 JTAG、Debug Hub (CLKA) 和调试核 (CLKB) 时钟应自由运行并保持稳定。

Debug Hub 插入准则

在某些情况下，设计包含的某个 Debug Bridge IP 实例可能看似阻止调试流程成功插入 Debug Hub，并且可能出现以下错误：

```
[Chipscope 16-336] Failed to find or create hub core for debug slave <debug
core name>. Insertion of debug hub is not supported when there are
instantiated debug bridge cores in either master mode or switch enabled in
the design. Either remove debug slave core or instantiate a debug bridge
master in the region of the debug slave
```

发生此问题的原因是调试流程检测到存在 Debug Bridge IP 并且不尝试自动插入 Debug Hub IP。但设计中的 Debug Bridge IP 实例并未处于正确模式下，故而无法连接到调试核。要解决此问题，请确保设计中至少有一个 Debug Bridge IP 处于 BSCAN-to-Debug Hub 模式下。

将 Vivado 调试核添加至 Dynamic Function eXchange 设计

Vivado 调试核可在 Dynamic Function eXchange 设计中进行例化，在可重配置模块中也同样如此。添加和连接这些核存在一些具体的要求和方法。请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909) 中的“使用 Vivado 调试核”章节，以了解添加和连接这些 Vivado 调试核所需的方法。

如需获取在 Dynamic Function eXchange 设计中例化调试核的示例和 Vivado 硬件管理器中该功能的相关描述信息，请参阅《Vivado Design Suite 教程：Dynamic Function eXchange》(UG947) 中的“Vivado 调试和 DFX 工程流程”章节。

第 13 章

在硬件中调试逻辑设计

设计中包含调试核后，您可使用运行时逻辑分析仪功能来对硬件中的设计进行调试。

使用 Vivado Logic Analyzer 进行设计调试

AMD Vivado™ Logic Analyzer 功能可用于与设计中运行的新 ILA、VIO 和 JTAG-to-AXI Master 调试核进行交互。要访问 Vivado Logic Analyzer 功能，请单击 Flow Navigator 的“Program and Debug”（烧录和调试）部分中的“Open Hardware Manager”（打开硬件管理器）按钮。

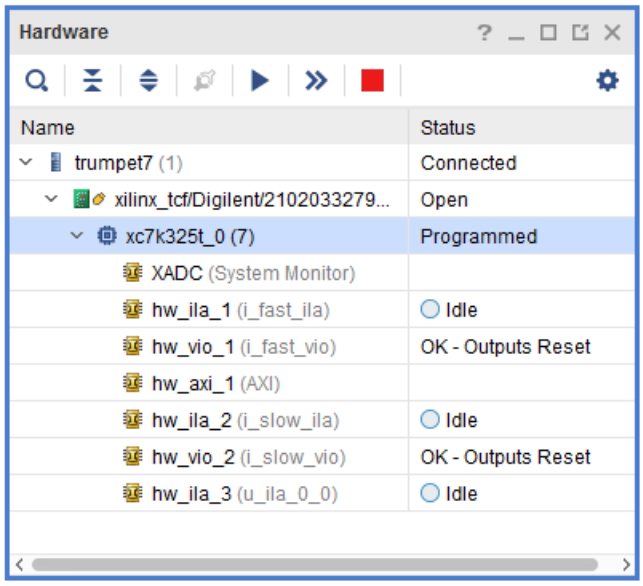
使用 ILA 调试核对硬件中的设计进行调试的步骤如下：

1. 连接到硬件目标并使用 `.pdi` 文件对 FPGA 或自适应 SoC 进行烧录。
2. 设置 ILA、调试核触发器和捕获控制。
3. 装备 ILA 调试核触发器。
4. 在“Waveform”（波形）窗口中查看从 ILA 调试核捕获的数据。
5. 使用 VIO 调试核来驱动控制信号和/或查看设计状态信号。
6. 使用 JTAG-to-AXI Master 调试核来运行传输事务，以便与设计中的各种 AXI 从核进行交互。

连接至硬件目标并执行器件烧录

在调试前对 FPGA 或自适应 SoC 进行烧录的步骤与“对 FPGA 或自适应 SoC 进行烧录”中所述的步骤完全相同。使用 `.pdi` 文件（包含新的 ILA、VIO 和 JTAG-to-AXI Master 调试核）烧录器件后，“Hardware”（硬件）窗口就会显示在扫描器件时检测到的调试核，并显示 RTL 实例名称（以括号括起）。

图 95：**显示调试核的“Hardware”窗口**



如需了解有关使用 ILA 核的更多信息，请参阅“设置 ILA 核以执行测量”。如需了解有关使用 VIO 核的更多信息，请参阅“设置 VIO 核以执行测量”。

重要提示！请确保 JTAG 时钟比调试核的时钟输入更慢。您可使用“Open New Hardware Target”Wizard（打开新硬件目标向导）或者使用以下 Tcl 命令来修改 JTAG 频率：`set_property PARAM.FREQUENCY 250000 [get_hw_targets */xilinx_tcf/Digilent/210203327962A]`

相关信息

器件烧录
设置 ILA 核以执行测量
设置 VIO 核以执行测量

Vivado 硬件管理器仪表板

Vivado 硬件管理器仪表板可帮助您管理 System Monitor（系统监控器）、ILA 和 VIO 调试核的各个窗口。这些仪表板支持您在自己的 AMD Vivado™ Design Suite 工程中创建、修改和保存这些窗口的配置。

默认仪表板

刷新硬件器件时如果检测到调试核，则将自动打开每个调试核的默认仪表板。

默认仪表板窗口

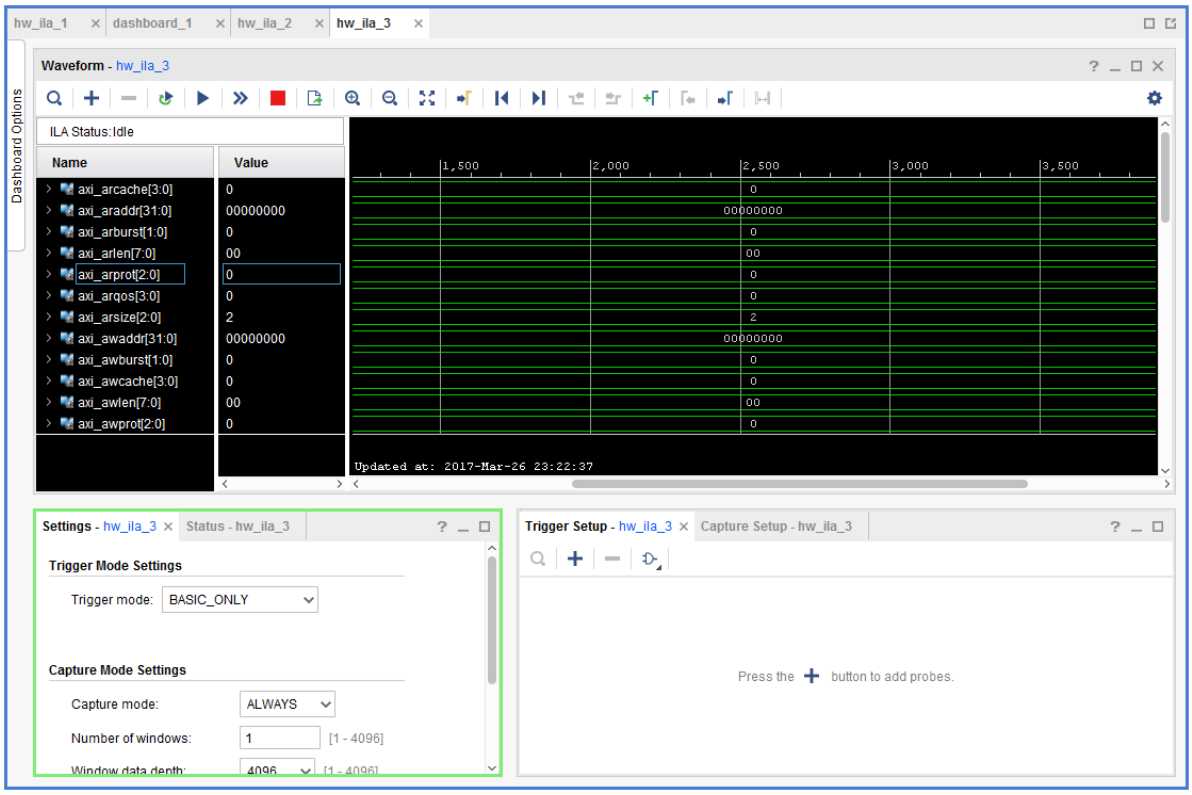
每个默认仪表板都包含与创建的仪表板所对应的调试核相关的窗口。为 ILA 调试核所创建的默认仪表板包含 5 个窗口：

- “Settings”（设置）窗口

- “Status”（状态）窗口
- “Trigger Setup”（触发器设置）窗口
- “Capture Setup”（捕获设置）窗口
- “Waveform”（波形）窗口

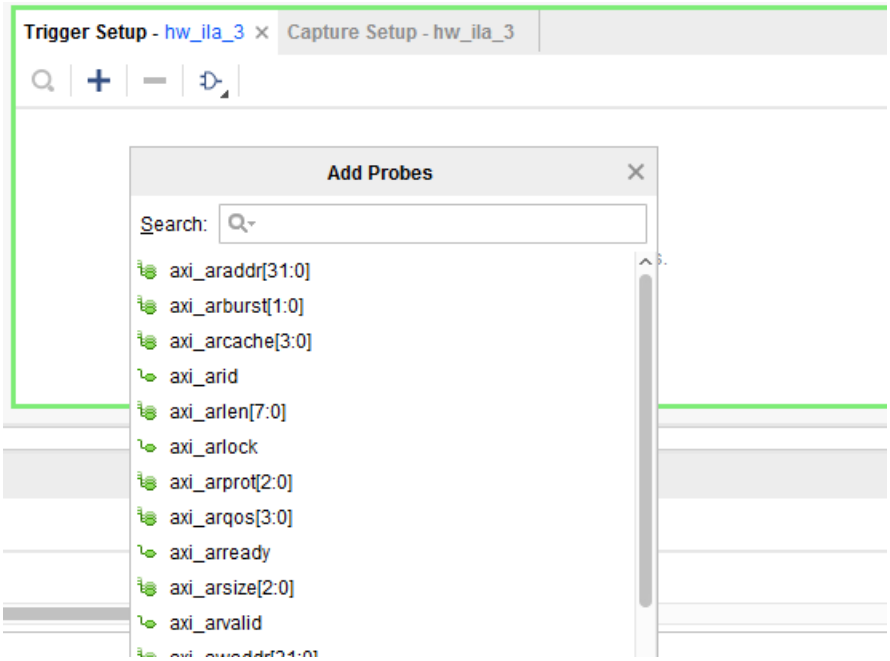
默认 ILA 仪表板示例如下所示：

图 96：**默认 ILA 仪表板**



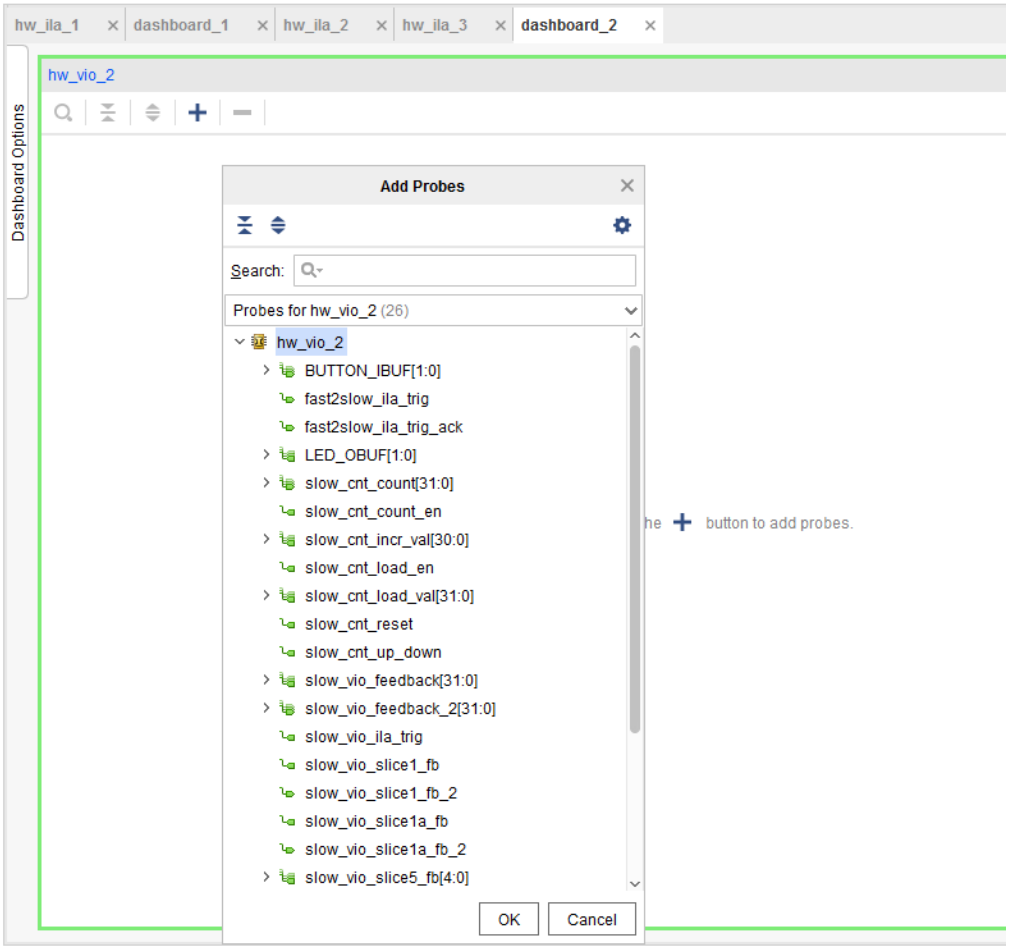
您可通过单击窗口中心的“+”按钮并从“Add Probes”（添加探针）窗口中选择探针，以开始向“Trigger Setup”（触发器设置）窗口添加探针，如下图所示：

图 97： **“Add Probes”窗口**



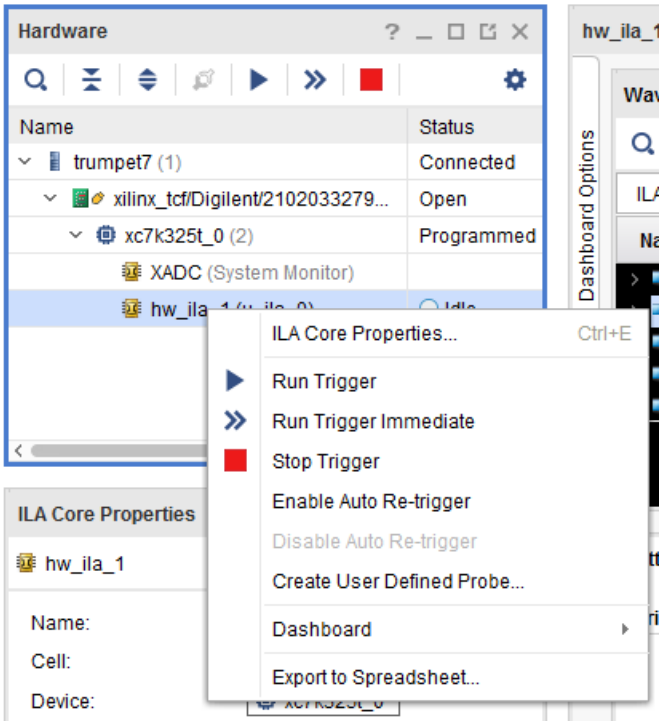
VIO 默认仪表板初始为空，您可向其中添加 VIO 探针，如下图所示。

图 98：**添加 VIO 探针**



要查看与调试核关联的仪表板，请在“Hardware”窗口中右键单击调试核对象、选择“Dashboard”选项，然后单击仪表板名称。双击“Hardware”（硬件）窗口中的调试核将弹出与该调试核关联的仪表板。

图 99：**关联仪表板**



仪表板内部的窗口控件

每个窗口都具有下列标题栏控件，以支持您操纵该窗口：

- 最小化
- 最大化
- 关闭

移动窗口

要移动窗口，请执行以下操作：

1. 选中窗口选项卡或标题栏然后拖动窗口。灰色边框表示移动后的窗口位置。
2. 要将窗口落实到位，请松开鼠标。

   注释：将某个窗口拖到另一个现有窗口上则会将这两个窗口选项卡放置在同一区域内。

重要提示！您无法将窗口移入或移出工作空间。但可在工作空间内调整窗口大小和移动窗口。

调整窗口大小

- 要调整窗口大小，请单击并拖动窗口边框。

  注释：光标定位于窗口边框时会变为调整大小光标或拖动手柄，表示您可单击并拖动窗口边框以调整窗口大小。
- 要展开窗口以查看整个环境，请单击窗口右上角的“Maximize”（最大化）按钮。
- 要将窗口复原至原始大小，请双击窗口标题栏或选项卡。

关闭窗口

- 要关闭窗口，请单击窗口右上角的“Close”（关闭）按钮。

  注释：在某些情况下，窗口选项卡中也包含此按钮。

- 右键单击窗口选项卡或标题栏，然后从弹出菜单中单击“Close”（关闭）。

窗口选项卡

每个窗口都包含对应选项卡，您可选中该选项卡以激活该窗口。选项卡位于某些窗口底部，例如，“Trigger Setup”（触发器设置）窗口和“Capture Setup”（捕获设置）窗口。

提示：要激活后一个选项卡，请按 Ctrl+Tab 键。要激活前一个选项卡，请按 Ctrl+Shift+Tab 键。要最大化或最小化窗口，请双击该窗口选项卡。

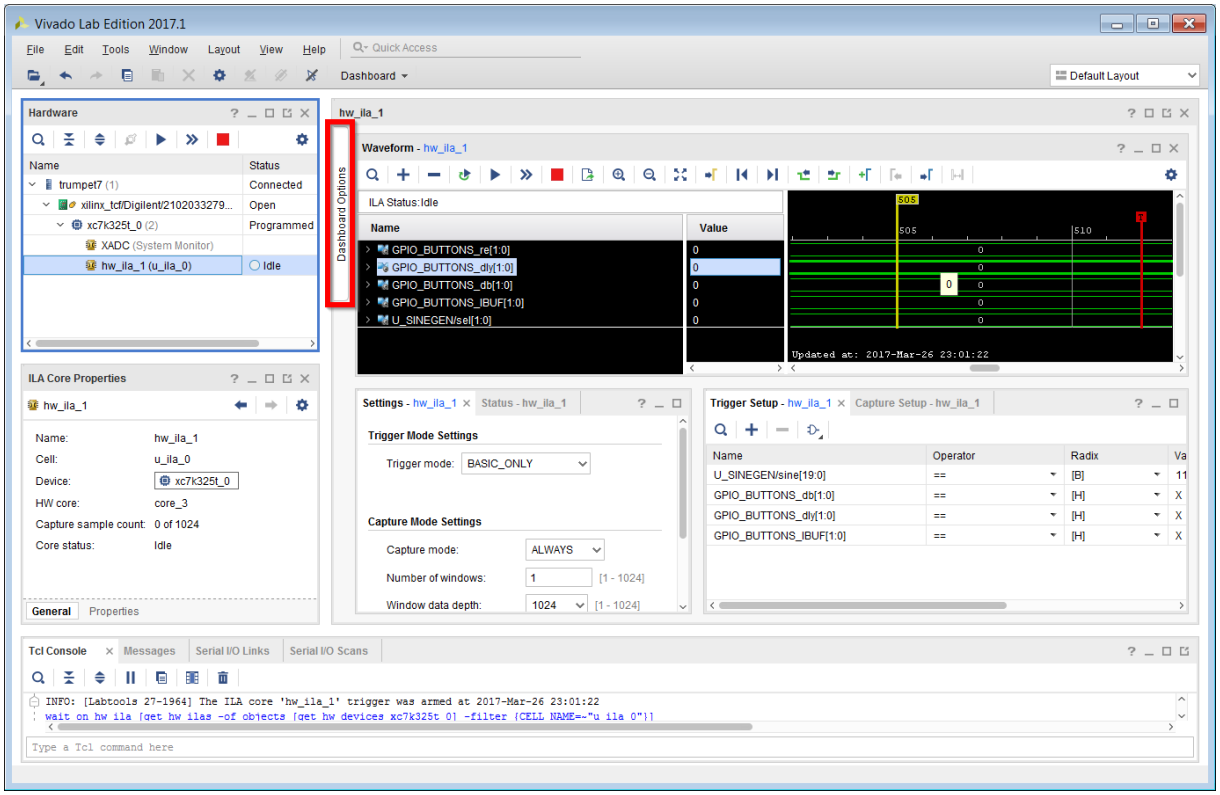
自定义仪表板

通常，默认仪表板中的窗口应足以供您调试设计和查看结果。但您可移动窗口（即，自定义仪表板）。例如，您可查看 ILA 状态和“Waveform”（波形）窗口，并在同一仪表板内控制 VIO 探针。在此类情况下，AMD 建议自定义仪表板以满足您的需求。

Dashboard Options

每个仪表板左侧都包含“Dashboard Options”（仪表板选项）滑出式菜单。请使用仪表板左侧的“Dashboard Options”按钮来打开其“Dashboard Options”设置。“Dashboard Options”设置允许您控制特定仪表板中显示的窗口。例如，您可以自定义 ILA 仪表板，使其同时包含某个 VIO 窗口。单击 VIO 窗口以将其包含在“Dashboard Options”中，这样此 VIO 窗口就会显示在 ILA 仪表板中，如下所示。现在，您即可添加自己感兴趣的 VIO 探针并触发 ILA 窗口。

图 100：**添加仪表板选项**

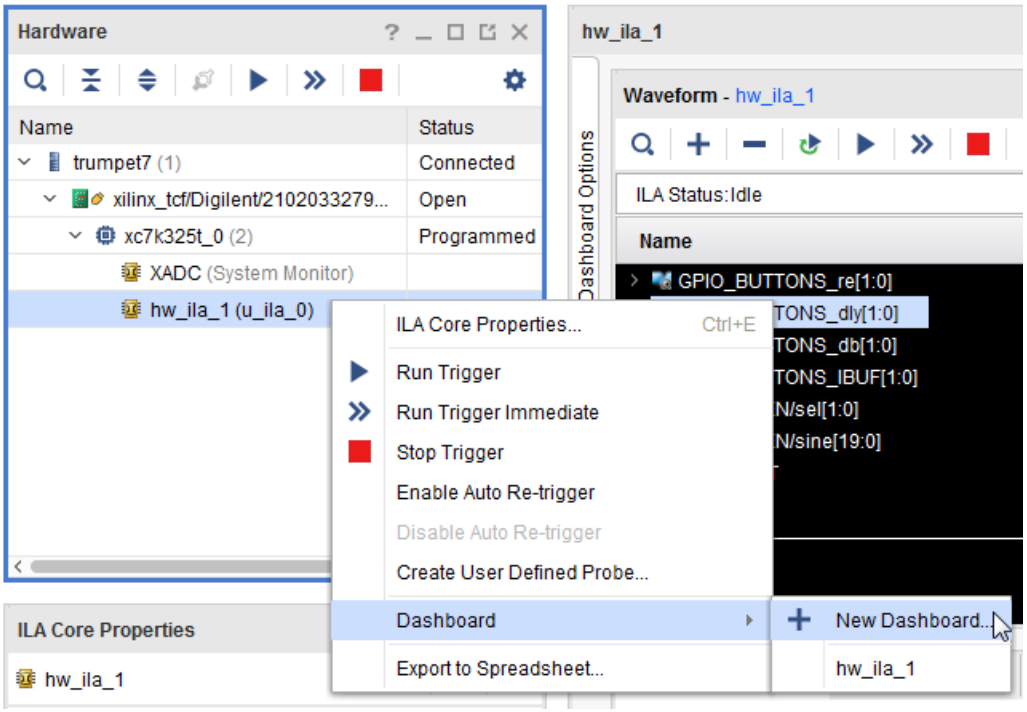


单击仪表板左侧的“Dashboard Options”按钮即可打开和关闭“Dashboard Options”滑出式菜单。

创建新的仪表板

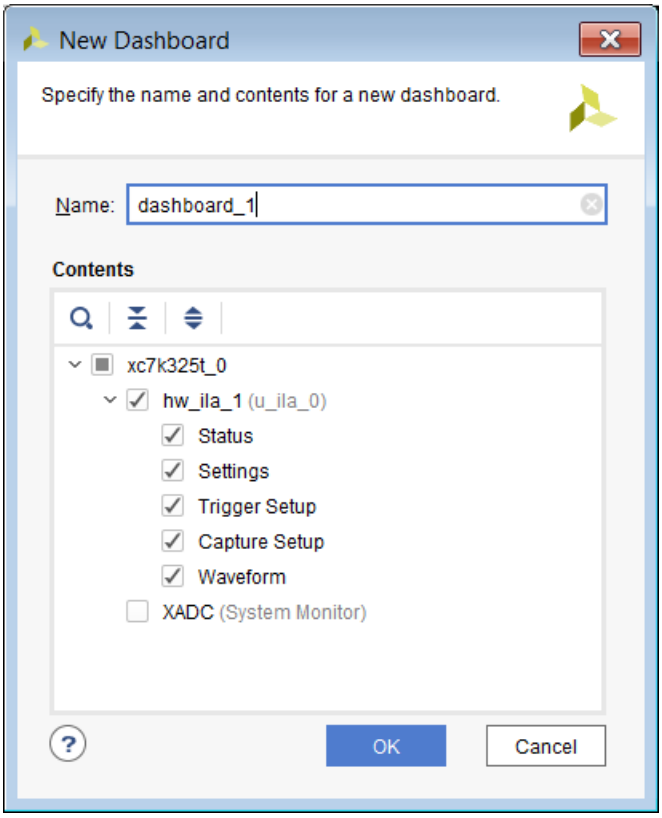
除了使用仪表板选项来自定义默认仪表板外，您还可以创建全新的仪表板。要创建新仪表板，请在“Hardware”（硬件）窗口中右键单击调试核对象并选择“Dashboard” → “New Dashboard”（仪表板 > 新建仪表板）选项，如下图所示。

图 101：**创建新的仪表板**



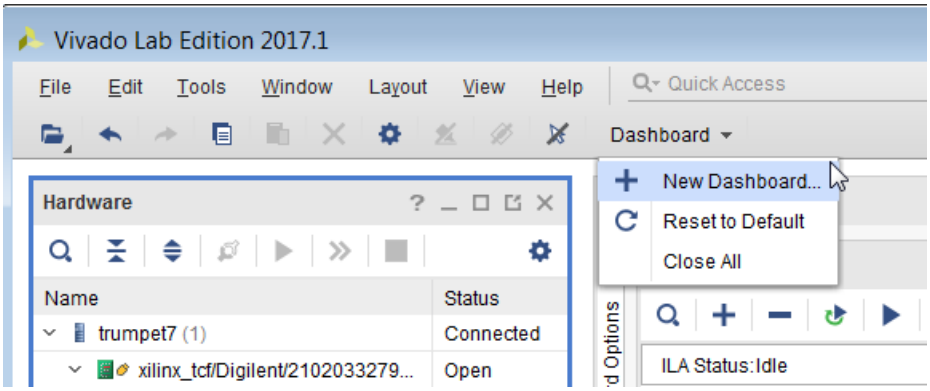
显示“New Dashboard”（新建仪表板）对话框时，您可根据需要对其进行自定义，然后单击“OK”。

图 102：**“New Dashboard”对话框**



您也可以使用仪表板工具栏按钮来创建新仪表板，如下所示：

图 103：**仪表板工具栏按钮**



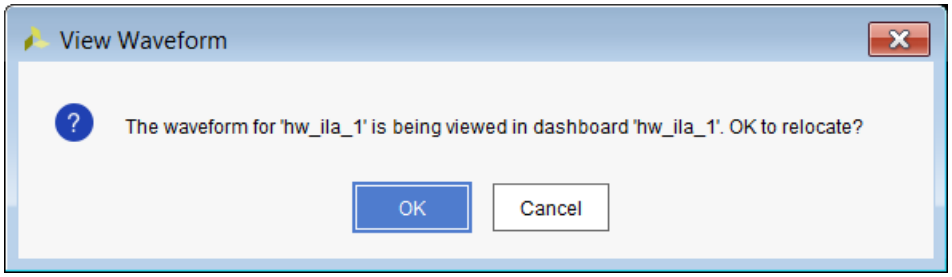
提示：要查看与调试核关联的所有仪表板，请右键单击“Hardware”（硬件）视图中的调试核，然后单击“Dashboard”（仪表板）。或者也可以双击“Hardware”视图中的调试核，这样将弹出与该调试核关联的仪表板列表。

提示：要使仪表板上的某个窗口变为浮动状态，AMD 建议创建一个含该窗口的仪表板，并使该仪表板变为浮动状态。

仪表板中的 ILA 波形窗口

每个 ILA 波形窗口都只能显示在单个仪表板中。如果您单击位于另一个仪表板中的“Waveform”（波形）窗口，那么将显示通知称此窗口位置已重定位，如下图所示。

图 104：**ILA 波形确认**



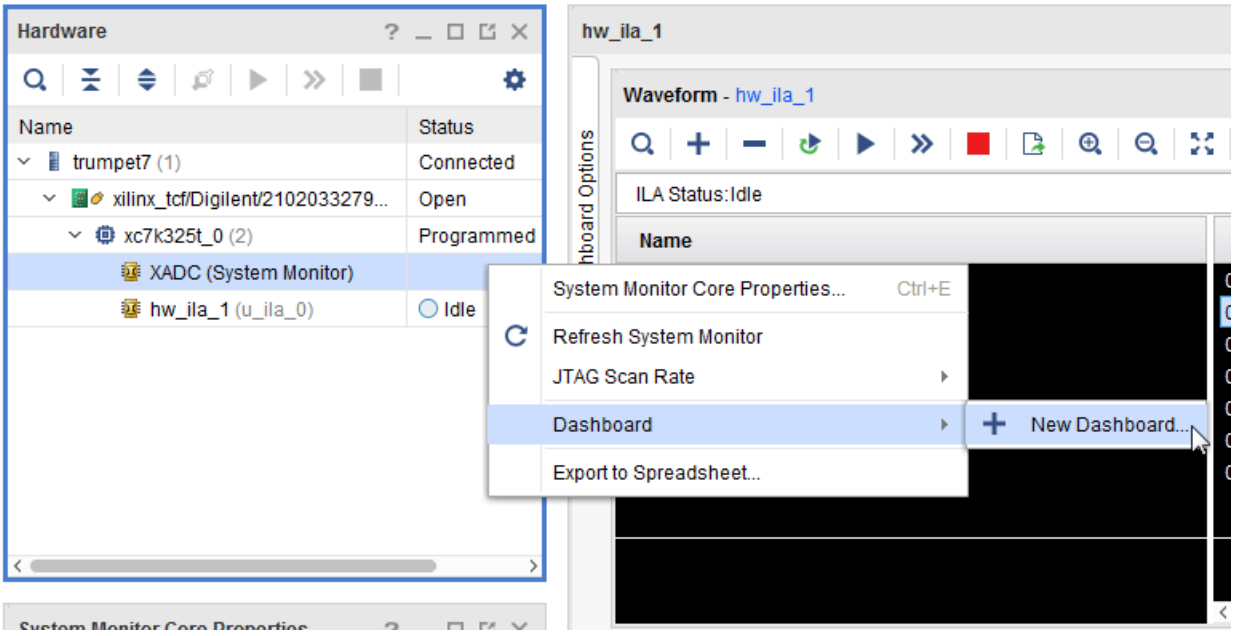
单击“OK”即可将“Waveform”窗口重定位至指定仪表板中。

提示：关闭“Waveform”窗口时请保存 ILA 数据。

系统监控器仪表板

您可将“XADC/System Monitor”（XADC/系统监控器）窗口包含在其自己的仪表板中，也可将其包含在另一个仪表板中。

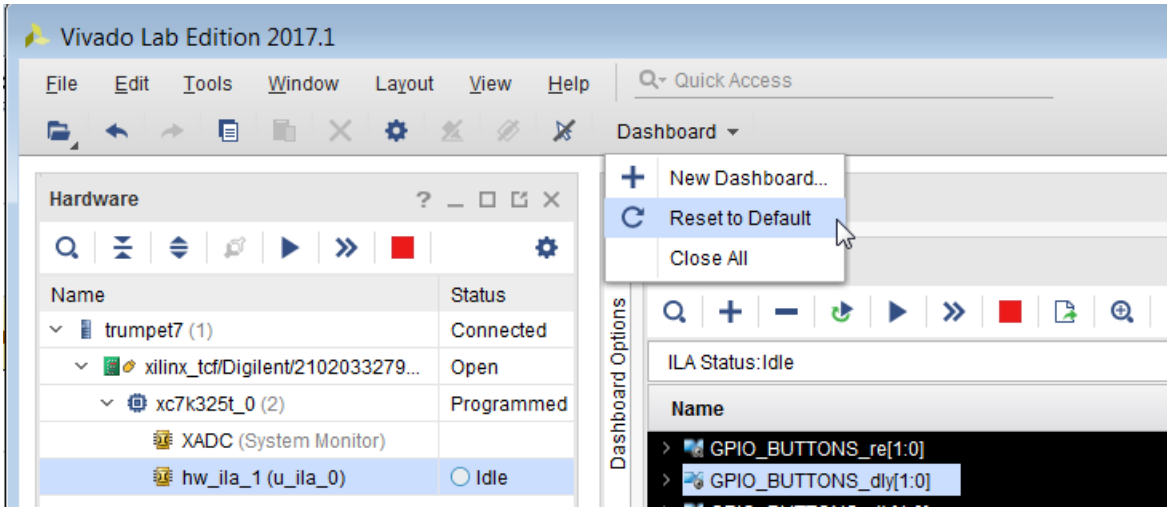
图 105：**系统监控器仪表板**



复位至默认仪表板

您可通过单击工具栏上的“Dashboard”（仪表板）并选择“Reset to Default”（复位至默认值）来将仪表板复位至默认状态。

图 106：**复位至默认仪表板**



关闭仪表板

您可通过单击工具栏上的“Dashboard”（仪表板）并单击“Close All”（全部关闭）来关闭所有仪表板。这样会删除所有仪表板及其中的用户设置。

您也可以通过单击单个仪表板的右上角“X”按钮来关闭该仪表板。这样会删除该仪表板及其中的所有用户设置。

保存用户仪表板首选项和设置

用户仪表板设置和首选项由 Vivado IDE 自动保存。关闭和重新打开工程时，用户设置和首选项将复原到硬件管理器中。

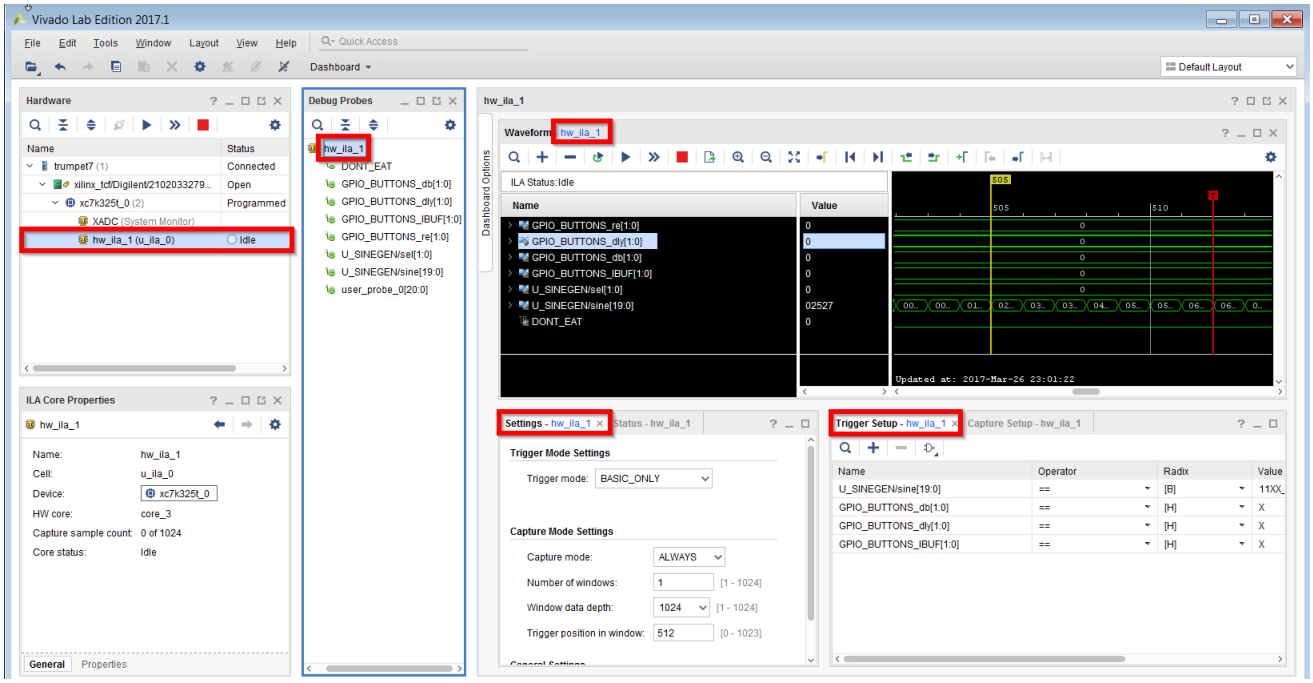
设置 ILA 核以执行测量

您添加到自己的设计中的 ILA 核会显示在“Hardware”（硬件）窗口中的目标器件下。如果未显示这些 ILA 核，请右键单击器件并选择“Refresh Device”（刷新器件）。这样会重新扫描 FPGA 或自适应 SoC 并刷新“Hardware”（硬件）窗口。

注释：如果烧录和/或刷新 FPGA 或自适应 SoC 后仍未显示 ILA 核，请检查并确保已使用正确的 `.bit` 文件完成器件烧录，并确认已实现的设计包含 ILA 核。此外，还请检查并确认有相应的 `.ltx` 探针文件（与 `.bit` 文件相匹配）与该器件关联。

单击 ILA 核（下图中名为 hw_ila_1 的核），以在“ILA Core Properties”（ILA 核属性）窗口中查看其属性。您可使用“Windows”→“Debug Probes”（窗口 > 调试探针）菜单选项来显示如下所示“Debug Probes”（调试探针）窗口，以便查看对应于 ILA 核的所有探针。

图 107：**各种视图中选中的 ILA 核**



添加探针

您可通过单击窗口工具栏或工作空间上的“+”按钮来将相关探针添加到 ILA 仪表板中的特定窗口。

写入调试探针信息

“Debug Probes”（调试探针）窗口包含有关您在自己的设计中使用 ILA 和/或 VIO 核探测的信号线的信息。此调试探针信息提取自您的设计，并存储在数据文件内，此数据文件通常带有 `.ltx` 文件扩展名。

通常，此调试探针文件是在实现进程中自动创建的。但是，您也可以使用 `write_debug_probes` Tcl 命令来将调试探针信息写出至文件：

1. 打开已综合的设计或网表设计。
2. 运行 `write_debug_probes` filename`.ltx` Tcl 命令。

> **重要提示！**如果使用非工程模式，则必须在执行 `opt_design` 命令后立即手动调用 `write_debug_probes` 命令。

读取调试探针信息

如果 Vivado IDE 处于工程模式下，并且在与器件关联的比特流烧录 (`.bit`) 文件所在目录中找到名为 `debug_nets.ltx` 的探针文件，那么调试探针文件将与硬件器件自动关联。

您还可以指定探针文件的位置：

1. 在“Hardware”（硬件）窗口中选择硬件器件。
2. 在“Hardware Device Properties”（硬件器件属性）窗口中设置探针文件位置。

3. 在“Hardware”窗口中，右键单击硬件器件并选择“Refresh Device”（刷新器件）以读取调试探针文件内容，将其与硬件器件中运行的设计中所提供的调试核加以关联并确认信息。

您还可使用以下 Tcl 命令来设置位置，以将名为 `C:\myprobes.ltx` 的调试探针文件与目标开发板上的首个器件进行关联：

```
% set_property PROBES.FILE {C:/myprobes.ltx} [lindex [get_hw_devices] 0] %
refresh_hw_device [lindex [get_hw_devices] 0]
```

重命名调试探针

您可使用“Debug Probes”（调试探针）窗口来重命名属于 ILA 或 VIO 核的调试探针。您可重命名调试探针，并将其添加到对应核的现有波形查看器中，或者可将其添加到 ILA 仪表板的各触发和/或捕获窗口中。这些名称可以是与调试探针关联的定制、长或短名称。

要执行这些操作，请右键单击 ILA/VIO 核的调试探针，并选择以下项之一：

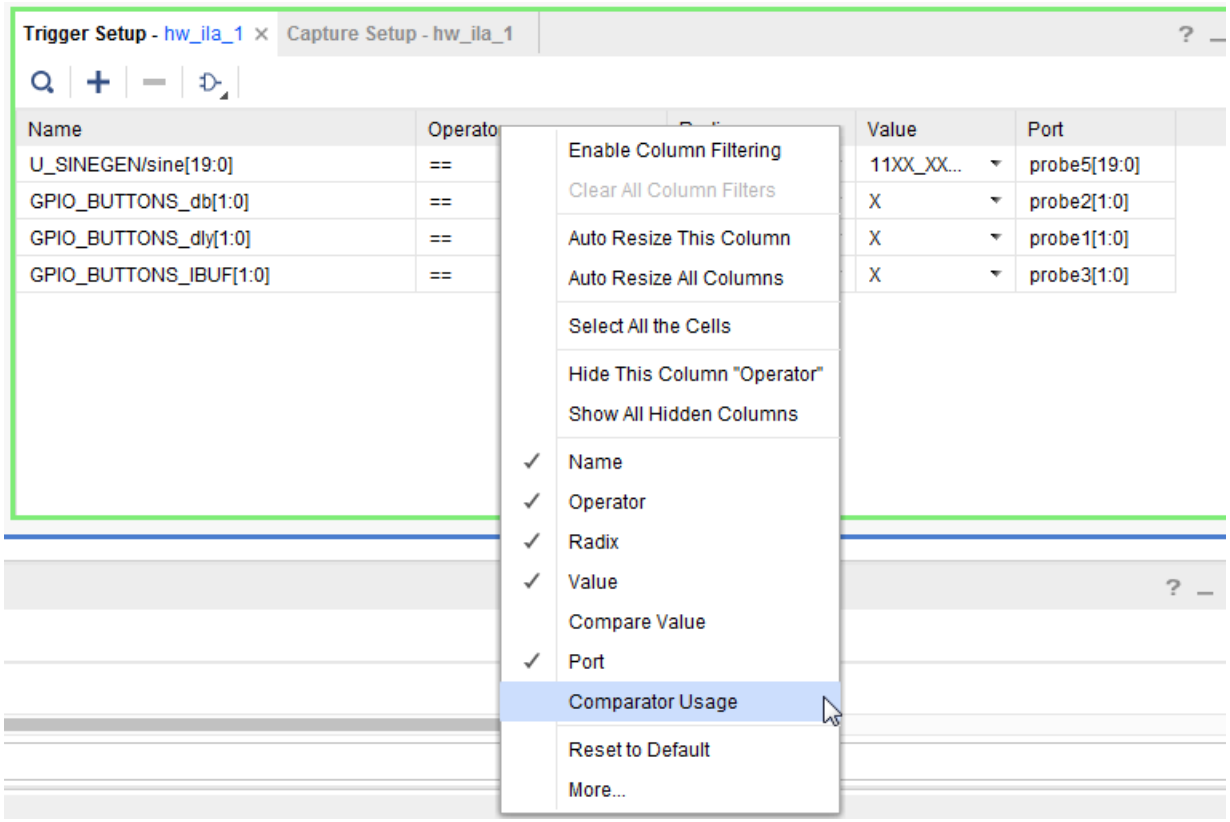
- “Rename”（重命名）：提示您将探针重命名为定制名称。
- “Name”（名称）：允许您为调试探针选择长、短或定制名称。Vivado IDE 窗口中后续引用的调试探针都会使用您所选的名称。
  - “Long”（长）：显示所探测的信号或总线的完整层级名称。
  - “Short”（短）：显示所探测的信号或总线的名称。
  - “Custom”（定制）：显示重命名时给予信号或总线的定制名称。

使用多个比较器

如果您已将探针和/或 ILA 调试核自定义为在“Basic”（基本）或“Advanced”（高级）模式下使用多个比较器，则可在“Basic Trigger Setup”（基本触发器设置）和“Advanced Trigger Setup”（高级触发器设置）窗口中使用这些比较器。

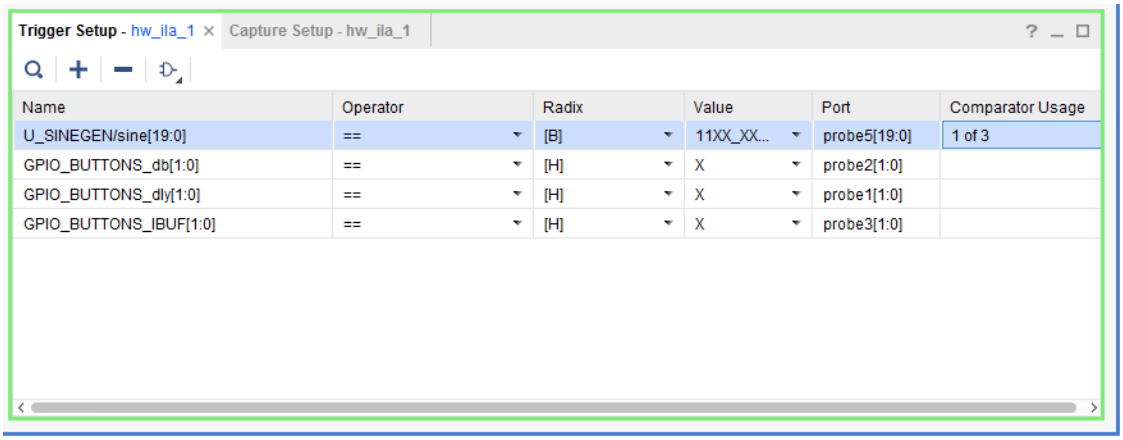
您可将探针添加到“Basic Trigger Setup”窗口中，并设置触发条件。“Comparator Usage”（比较器使用情况）列可根据特定比较条件（超出此探针关联的比较器总数范围），提供有关探针内所用的比较器的信息。

图 108：**Basic Trigger Setup - Comparator Usage**



提示： “Comparator Usage”列为隐藏列。要启用此列，请右键单击“Trigger Setup”（触发器设置）列标题（如下所示），然后单击“Comparator Usage”（比较器使用情况）。

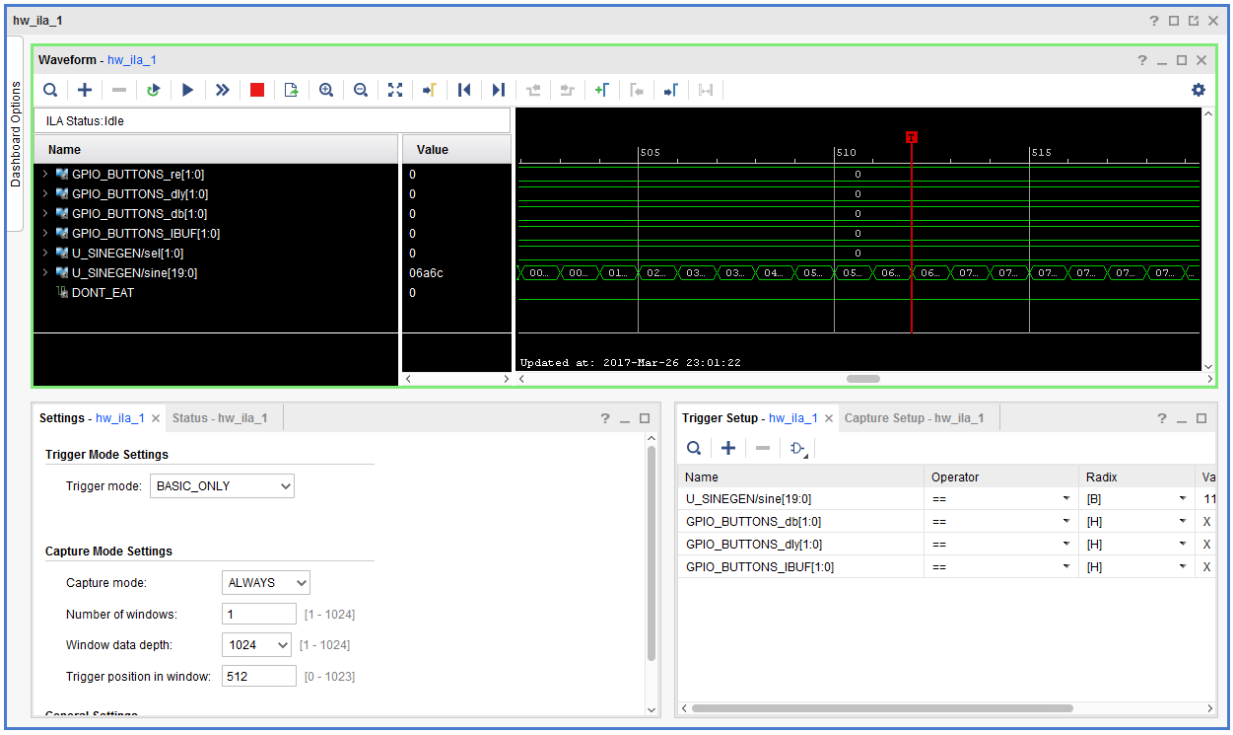
图 109：**“Comparator Usage”列**



使用 ILA 默认仪表板

ILA 仪表板（请参阅下图）是给定 ILA 核相关的所有状态和控制信息的集中显示位置。刷新硬件器件并首次检测到 ILA 核时，将自动打开该核的默认 ILA 仪表板。如果需要手动打开或重新打开此仪表板，请在“Hardware”（硬件）窗口中右键单击 ILA 核对象，然后单击“Default Dashboard”（默认仪表板）。

图 110：**ILA 仪表板**



您可使用 ILA 仪表板通过多种方式来与 ILA 调试核进行交互：

- 设置触发器模式，以便在硬件中发生各种事件时触发：
  - BASIC_ONLY：“ILA Basic Trigger Mode”（ILA 基本触发器模式）可用于在满足调试核比较结果的基本 AND/OR 功能时触发 ILA 核。
  - ADVANCED_ONLY：“ILA Advanced Trigger Mode”（ILA 高级触发器模式）可用于按用户定义的状态机指定的条件来触发 ILA 核。
  - TRIG_IN_ONLY：“ILA TRIG_IN Trigger Mode”（ILA TRIG_IN 触发器模式）可用于在 ILA 核的 TRIG_IN 管脚从低位转换至高位时触发 ILA 核。
  - BASIC_OR_TRIG_IN：“ILA BASIC_OR_TRIG_IN Trigger Mode”（ILA BASIC_OR_TRIG_IN 触发器模式）可用于在 ILA 核的 TRIG_IN 管脚执行逻辑 OR 操作并且目标模式为 BASIC_ONLY 触发器模式时触发 ILA 核。
  - ADVANCED_OR_TRIG_IN：“ILA ADVANCED_OR_TRIG_IN Trigger Mode”（ILA ADVANCED_OR_TRIG_IN 触发器模式）可用于在 ILA 核的 TRIG_IN 管脚执行逻辑 OR 操作并且目标模式为 ADVANCED_ONLY 触发器模式时触发 ILA 核。
- 设置触发器输出模式。
- ALWAYS 和 BASIC 捕获模式可用于控制要捕获的数据的筛选操作。
- 设置 ILA 捕获窗口的数量。

- 设置 ILA 捕获窗口的数据深度。
- 将触发器位置设置为捕获窗口内的任意样本。
- 监控 ILA 调试核的触发和捕获状态。

用户定义的调试探针

在硬件管理器中使用用户定义的调试探针（也称为 hw_probes）即可利用物理 ILA 探针端口与常量值的组合来创建探针。您可在硬件管理器的“Trigger Setup”（触发器设置）窗口或“Waveform”（波形）窗口中使用这些探针。这些探针创建成功后，将列在“Debug Probes”（调试探针）窗口中，并包含在创建期间与之关联的调试核内。

您可创建的用户定义的探针类型如下：

- ILA 探针端口。
- 1 个或多个常量值（0 和/或 1）。
- ILA 探针端口与常量值混合。

重要提示！包含常量值的用户定义的探针只能在“Waveform”窗口中使用。这些探针无法在“Trigger Setup”窗口中使用。

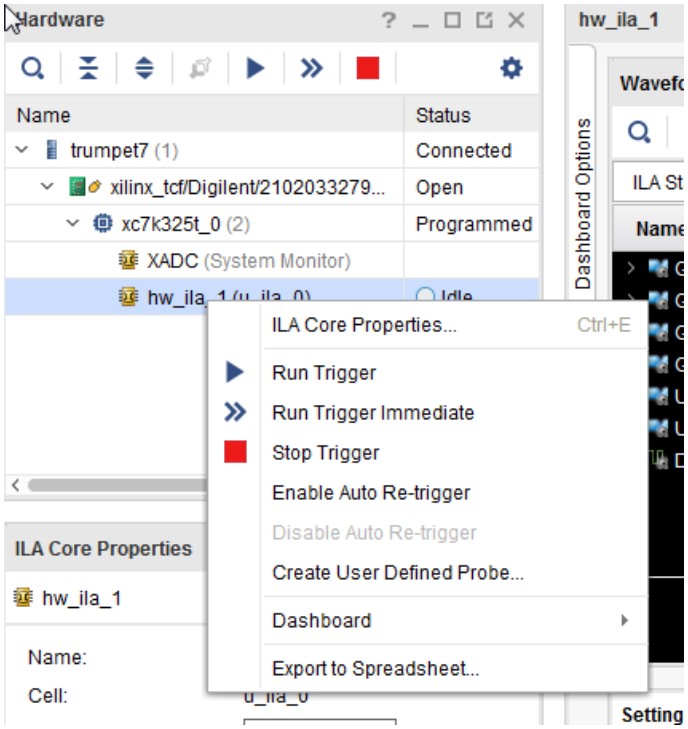
提示：您只能在 ILA 调试核上创建用户定义的探针。当前不支持为 VIO 核创建用户定义的调试探针。

创建用户定义的调试探针

GUI 流程

要在 Vivado IDE 硬件管理器中创建用户定义的调试探针，请在“Hardware”窗口中，右键单击要探测的 ILA 核并选中“Create User Defined Probe”（创建用户定义的探针）。

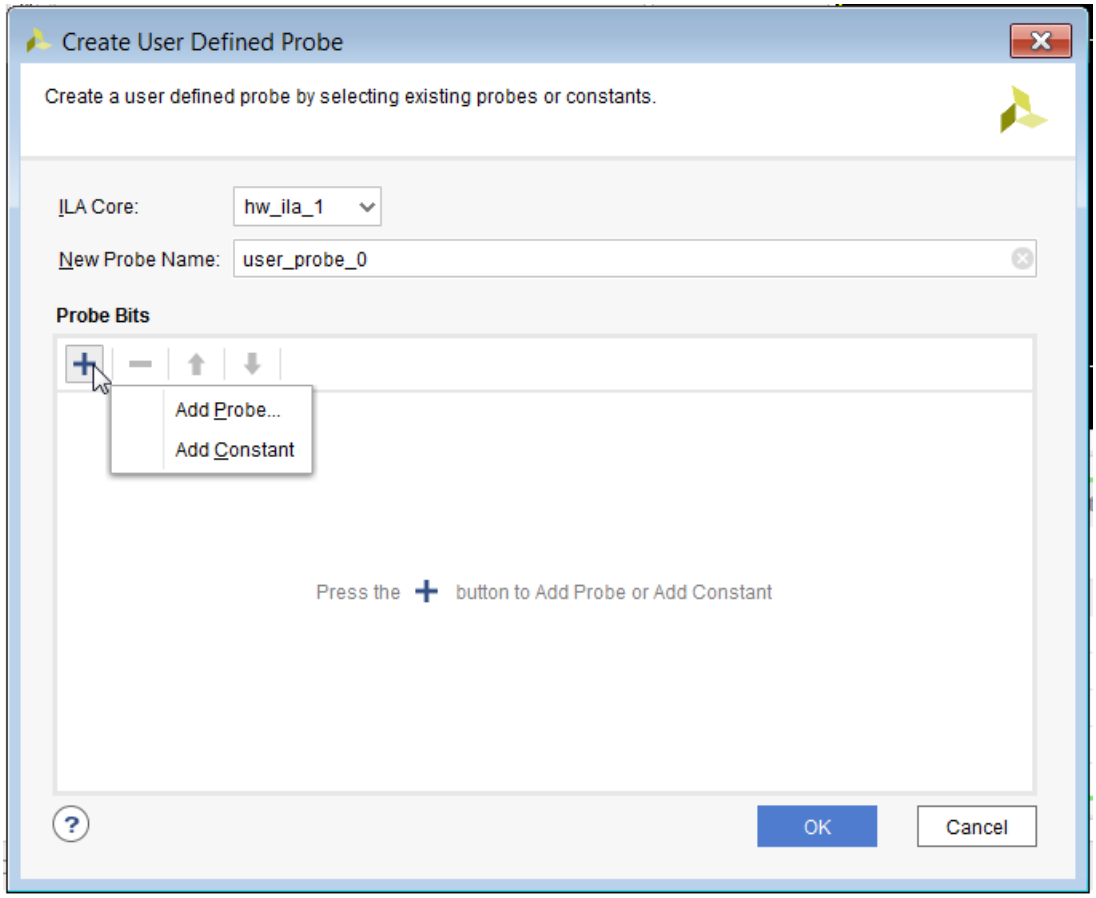
图 111：**选择“Create User Defined Probe”**



这样可启动“Create User Defined Probe”对话框。选择要在其中创建探针的 ILA 核、新探针的名称，最后选择探针位和/或组成此新探针的常量。

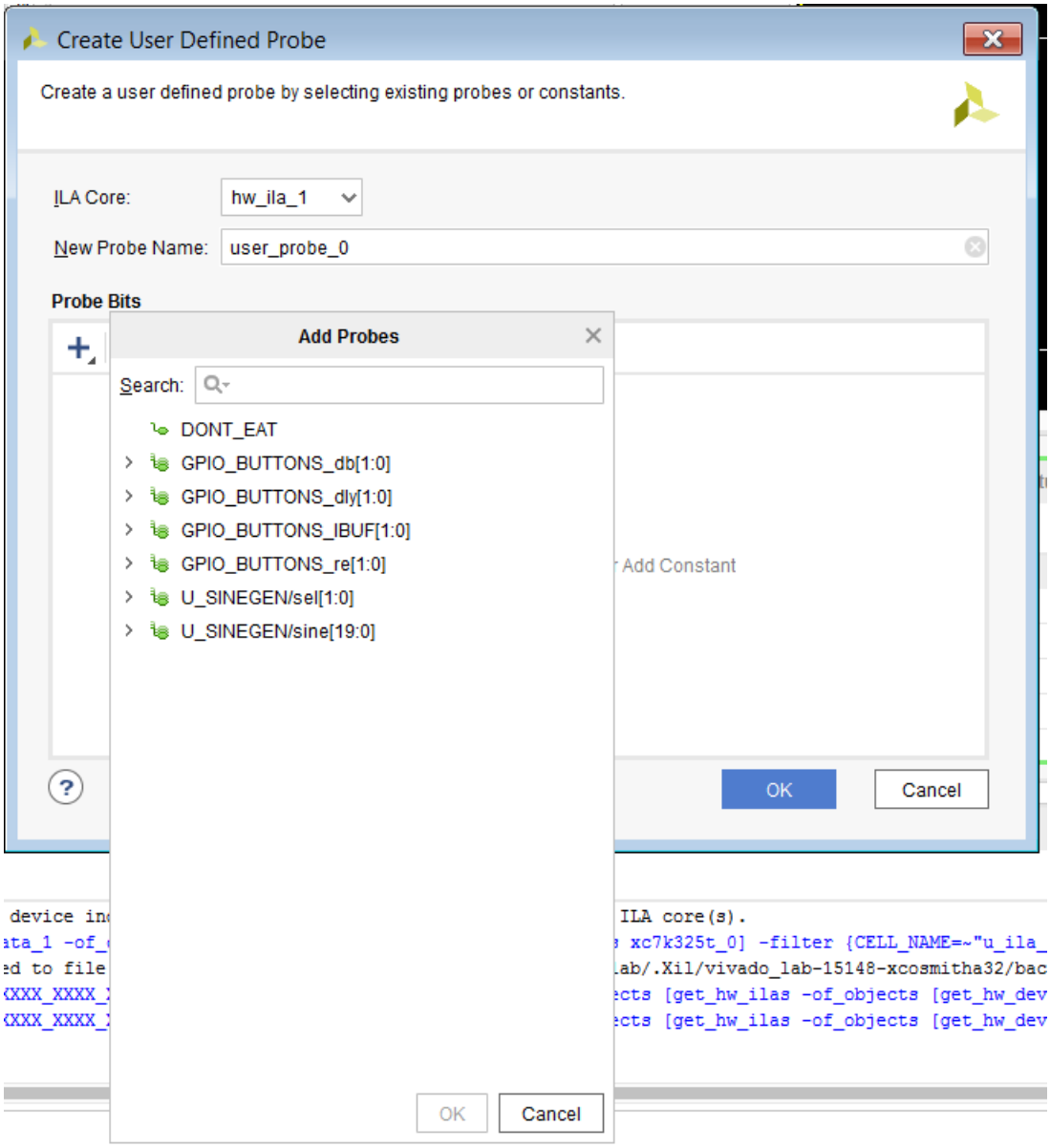
要向此新探针添加特定探针位，请单击“+”按钮，并选择“Add Probe”（添加探针）。

图 112：**“Create User Defined Probe”对话框**



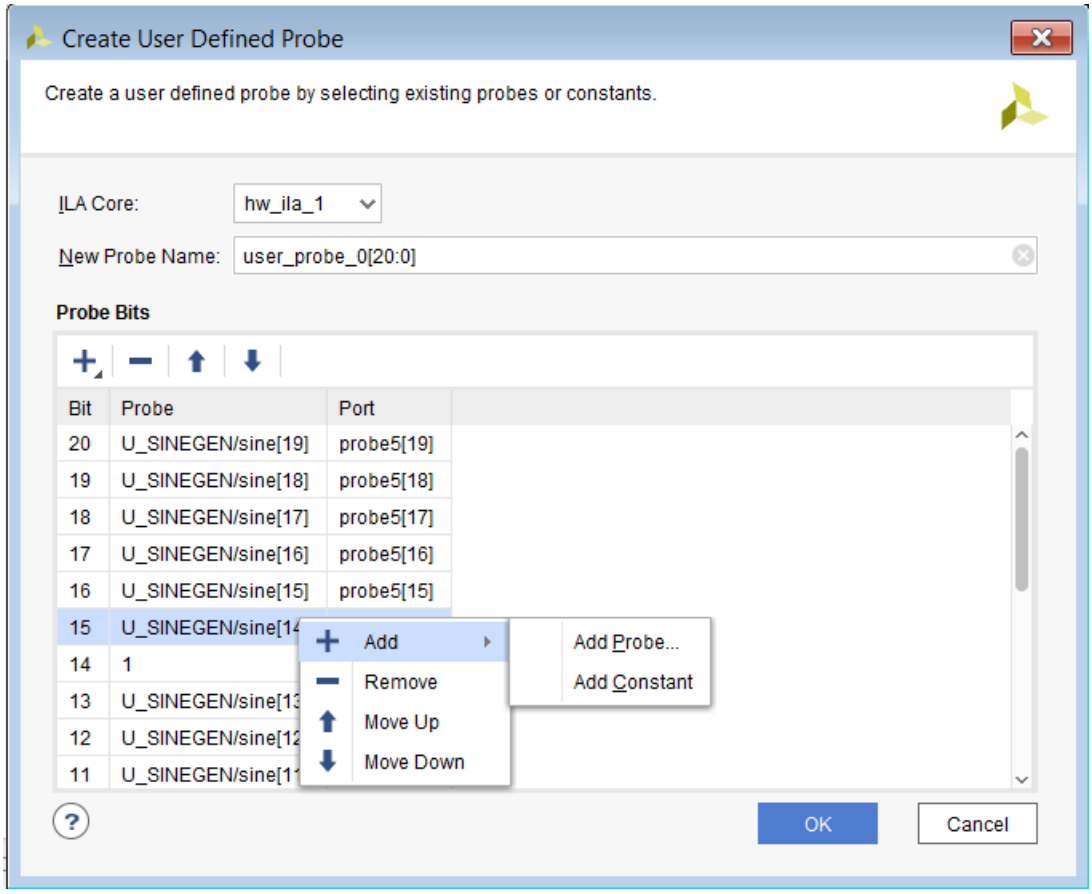
这样将启动“Add Probes”（添加探针）对话框，以便您选择现有探针或者现有探针的特定位。

图 113：**“Add Probes”对话框**



您还可以在“Create User Defined Probe”对话框中添加或移除位。将特定位上移或下移，如下图所示。

图 114：**在“Create User Defined Probe”对话框中编辑位**



Tcl 流程

要创建用户定义的调试探针，请使用 `create_hw_probe` Tcl 命令。

```
create_hw_probe [-verbose] [-map <arg>] <name> <core>
```

其中：

- name：是 `hw_probe` 的名称。对于属于同一器件的 `hw_probes`，此名称必须唯一。必须指定总线探针的范围。例如，`myNewProbe[31:0]`。
- core：是要与探针关联的 `hw_ila`。
- -map：表示声明的位元，这些位元将作为用户定义的探针的基础。

创建用户定义的调试探针的示例：

```
# Given a 512-bit counter "counterA[511:0]": Connects [255:223] to #    ILA probe port 0 [31:0] # Create a user-defined probe called foobar pointing at the #    ILA buffer specified range [255:223]     create_hw_probe -map {probe0[31:0]} {foobar [255:223]} [get_hw_ilas hw_ila_1] # Constant only probe. NO triggering allowed on constant ONLY probes.     create_hw_probe -map {0} {my_constant_gnd[0:0]} [get_hw_ilas hw_ila_1] # Create a user-defined probe as a mix of constants and ILA probe ports     create_hw_probe -map {0000 probe0[31:0] 1010} {my_mixed_probe[47:8]} [get_hw_ilas hw_ila_1]
```

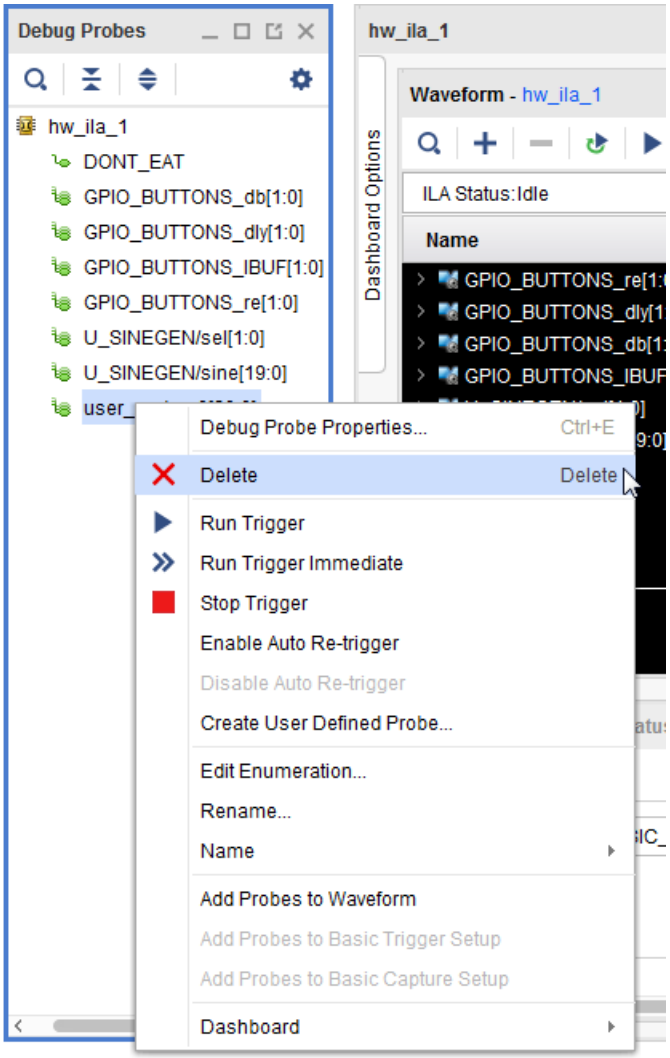
```
# Creating scalar hw_probe called "foobar" from probe1: create_hw_probe -
map {probe1} foobar [get_hw_ilas hw_ila_1] # Creating scalar hw_probe
called "foobar" from bit 3 of probe0: create_hw_probe -map {probe0[3]}
foobar [get_hw_ilas hw_ila_1] # Creating vector hw_probe called
"foobar[0:0]" from probe1: create_hw_probe -map {probe1} foobar[0:0]
[get_hw_ilas hw_ila_1] # Creating vector probe called "foobar[3:0]" from
probe0: create_hw_probe -map {probe0} foobar[3:0] [get_hw_ilas hw_ila_1] #
Creating vector probe called "foobar[3:2]" from probe0[1:0]:
create_hw_probe -map {probe0[1:0]} foobar[3:2] [get_hw_ilas hw_ila_1]
```

删除用户定义的调试探针

GUI 流程

要在 Vivado IDE Hardware Manager 中删除用户定义的探针，请依次选中“Window” → “Debug Probe”（窗口 > 调试探针）。这样会在“Hardware Manager”（硬件管理器）仪表板旁打开“Debug Probes”（调试探针）窗口。右键单击此窗口中的相应探针，然后单击“Delete”（删除），如下所示：

图 115：**删除调试探针**



Tcl 流程

您可使用 `delete_hw_probe` Tcl 命令删除用户定义的调试探针。

例如，要删除先前使用 `create_hw_probe` 创建的 `foobar` 探针，请执行以下操作：

```
delete_hw_probe [get_hw_probes foobar -of_objects [get_hw_ilas -of_objects
[get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
```

用户定义的调试探针的持久性

在工程流程中，硬件管理器会自动保留其中创建的任何用户定义的探针。下次打开该工程并使用 Vivado 硬件管理器连接到硬件目标时，这些用户定义的探针就会恢复。如果这些用户定义的调试探针可在“Basic triggering”（基本触发）窗口中使用，或者已添加到“Waveform”（波形）窗口中，那么在打开工程并连接到硬件管理器中的目标时，所有窗口中显示的探针设置都与先前关闭工程时的设置完全相同。

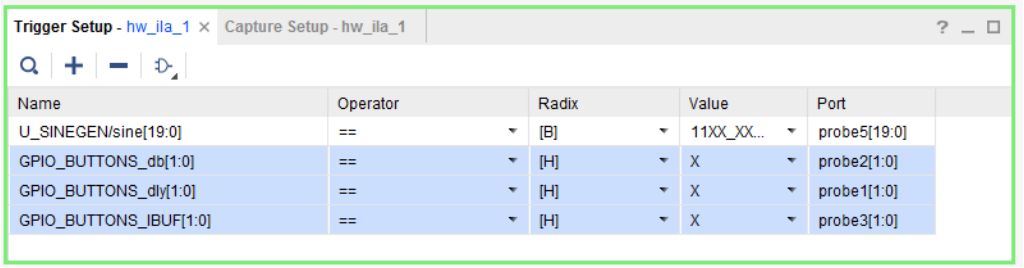
与用户定义的探针进行交互

硬件管理器中创建的任意用户定义的调试探针均可在“Basic triggering”（基本触发）窗口、“Advanced Triggering”（高级触发）窗口和/或“Waveform”（波形）窗口中使用。唯一例外是涉及常量值的用户定义的调试探针。这些调试探针只能在“Waveform”（波形）窗口中使用。

使用基本触发器模式

基本触发器模式用于描述触发条件，即由参与其中的调试探针比较器组成的全局布尔公式。当“Trigger Mode”（触发器模式）设置为 BASIC_ONLY 或 BASIC_OR_TRIG_IN 时，即启用基本触发器模式。使用“Basic Trigger Setup”（基本触发器设置）窗口（请参阅下图）来创建此触发条件和调试探针比较值。

图 116：**ILA 基本触发器设置窗口**



您也可以使用 `set_property` Tcl 命令来更改 ILA 核的触发模式。例如，要将 ILA 核 `hw_ila_1` 的触发模式更改为“BASIC_ONLY”，请使用以下命令：

```
set_property CONTROL.TRIGGER_MODE BASIC_ONLY [get_hw_ilas hw_ila_1]
```

在“Basic Trigger Setup”窗口中添加探针

使用基本触发器模式的第一步是判定要将哪些 ILA 调试探针添加到触发条件中。具体操作是从“Debug Probes”（调试探针）窗口中选中所需 ILA 调试探针，然后右键单击并选择“Add Probes to Basic Trigger Setup”（在基本触发器设置中添加探针），或者将探针拖放到“Basic Trigger Setup”（基本触发器设置）窗口中。

注释：您可将首个探针拖放到“Basic Trigger Setup”窗口中的任意位置，但必须将第二个以及后续每个探针拖放到第一个探针上层。新探针始终添加到表中前一个添加的探针上层。您也可以按此方式使用拖放操作来对表中的探针进行重新排列。

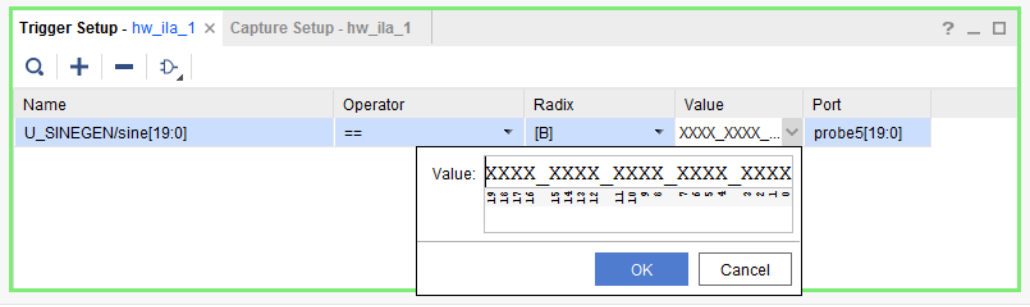
重要提示！仅限包含在“Basic Trigger Setup”窗口中的探针方可参与触发条件。不包含在此窗口中的任何探针均设置为“忽略”值，并且不会被包含在触发条件中以供使用。

要从“Basic Trigger Setup”窗口中移除探针，请执行以下任一操作：选中探针并按“Delete”（删除）键，或者右键单击并选中“Remove”（移除）选项。

设置基本触发器比较值

ILA 调试探针触发器比较器可用于检测 ILA 核的探针输入上的等于或不等于条件。触发条件即对每个 ILA 探针触发器比较器结果执行布尔“AND”、“OR”、“NAND”或“NOR”计算的结果。要为给定 ILA 探针指定比较值，请在“Basic Trigger Setup”（基本触发器设置）窗口中选中给定 ILA 调试探针的“Value”（值）单元以将其打开（请参阅下图）。

图 117：**ILA 探针比较值对话框**



提示：更改“Radix”（基数）前，请确保该值已设置为适用于新基数的值。

ILA 探针比较值设置

“Basic Trigger Setup”（基本触发器设置）窗口包含 3 个单元，可供您在与每个探针逐一对应的特定行中进行配置：

1. “Operator”（运算符）：此比较运算符可供您设置为以下值：
   - ==（等于）
   - !=（不等于）
   - <（小于）
   - <=（小于或等于）
   - >（大于）
   - >=（大于或等于）
2. “Radix”（基数）：此基数或基值可供您设置为以下值：
   - [B] 二进制
   - [H] 十六进制
   - [O] 八进制

- [U] 无符号十进制
- [S] 有符号十进制

3. “Value”（值）：该比较值通过使用运算符与设计中信号线上的实时值进行比较，这些信号线已连接到 ILA 调试核的探针输入。根据 Radix 设置，Value 字符串如下：
   - 二进制
     - 0：逻辑 0
     - 1：逻辑 1
     - X：忽略
     - R：上升或者从低到高转换
     - F：下降或从高到低转换
     - B：从低到高转换或从高到低转换
     - N：无转换（当前样本值与先前值相同）
   - 十六进制
     - X：对应于 Value 字符串字符的所有位均为“忽略”的值
     - 0-9：值 0 到 9
     - A-F：值 10 到 15
   - 八进制
     - X：对应于 Value 字符串字符的所有位均为“忽略”的值
     - 0-7：值 0 到 7
   - 无符号十进制
     - 任意非负整数值
   - 有符号十进制
     - 任意整数值

设置基本触发条件

您可使用“Basic Trigger Setup”（基本触发器设置）窗口左侧的工具栏按钮来设置触发条件，此窗口上有一个逻辑门形状的图标（请参阅下图）。您还可使用 `set_property` Tcl 命令来更改 ILA 核的触发条件：

```
set_property CONTROL.TRIGGER_CONDITION AND [get_hw_ilas hw_ila_1]
```

下表中显示了 4 个可能的值的含义。

图 118：**设置基本触发条件**

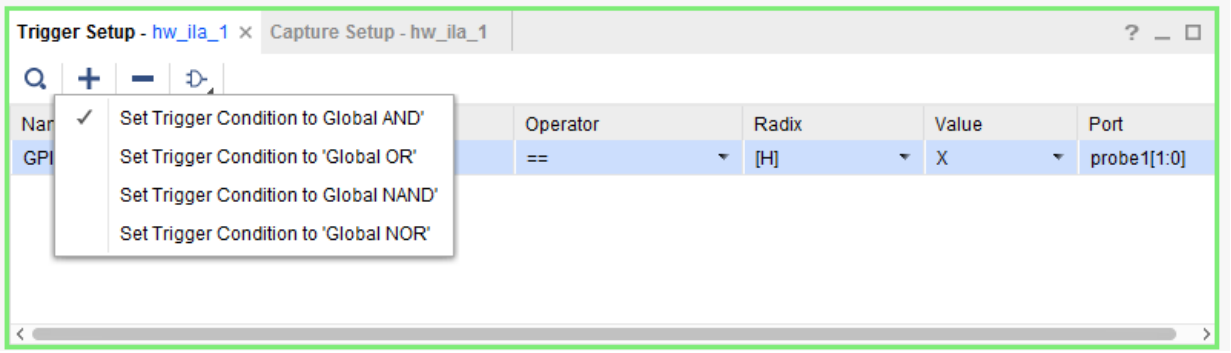


表 9：**基本触发条件设置描述**

GUI 中的触发条件设置	CONTROL.TRIGGER_CONDITION 属性值	触发条件输出
Global AND	AND	如果所有参与的探针比较器求值结果均为“true”，那么触发条件为“true”，否则触发条件为“false”。
Global OR	或	如果至少一个参与的探针比较器求值结果为“true”，那么触发条件为“true”，否则触发条件为“false”。
Global NAND	NAND	如果至少一个参与的探针比较器求值结果为“false”，那么触发条件为“true”，否则触发条件为“false”。
Global NOR	NOR	如果所有参与的探针比较器求值结果均为“false”，那么触发条件为“true”，否则触发条件为“false”。

重要提示！如果 ILA 核包含 2 个或 2 个以上的调试探针，且这些调试探针串联在一起并共享 ILA 核的单个物理探测端口，那么仅支持“Global AND”(AND) 和“Global NAND”(NAND) 触发条件设置。由于探测端口比较器逻辑所限，不支持“Global OR”(OR) 和“Global NOR”(NOR) 函数。如果要使用“Global OR”(OR) 或“Global NOR”(NOR) 触发条件设置，请确保将每个唯一的信号线或总线信号线分配到 ILA 核的不同探测端口。

使用高级触发器模式

ILA 核可在核生成时或插入时进行配置，以包含下列高级触发器功能：

- 含最多 16 种状态的触发器状态机。
- 每个状态均包含单向、双向或三向条件分支。
- 在触发器状态机程序中最多可使用 4 个计数器来保留多个事件的追踪记录。
- 在触发器状态机程序中最多可使用 4 个标志来表示某些分支的发生时间。
- 此状态机可执行“goto”、“trigger”及各种计数器和标志相关的操作。

如果硬件器件内运行的设计中的 ILA 核包含高级触发器功能，那么可通过将“ILA Dashboard”（ILA 仪表板）的“ILA Properties”（ILA 属性）窗口中的“Trigger mode”（触发器模式）控制设置为 ADVANCED_ONLY 或 ADVANCED_OR_TRIG_IN 来启用高级触发器模式功能。

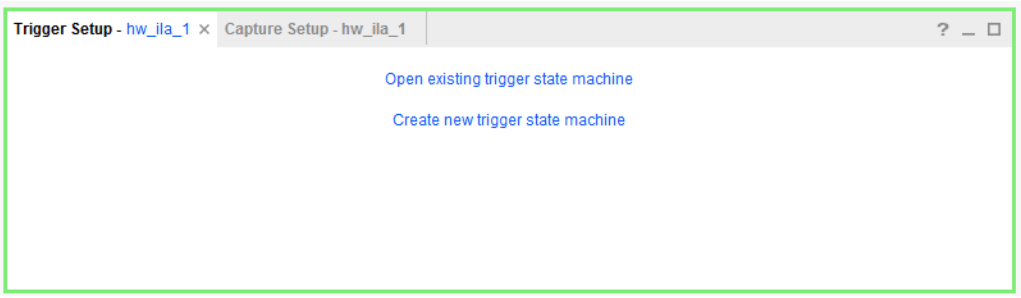
指定触发器状态机程序文件

将“Trigger”（触发器）模式设置为 ADVANCED_ONLY 或 ADVANCED_OR_TRIG_IN 时，在 ILA 仪表板中会发生 2 项更改：

1. 在“ILA Properties”（ILA 属性）窗口中会显示名为 Trigger State Machine 的新控件。
2. “Basic Trigger Setup”（基本触发器设置）窗口将被替换为“Trigger State Machine code editor”（触发器状态机代码编辑器）窗口。

如果您首次指定 ILA 触发器状态机程序，那么将显示“Trigger State Machine Code Editor”（触发器状态机代码编辑器）窗口，如下图所示。

图 119：**创建或打开触发器状态机程序文件**

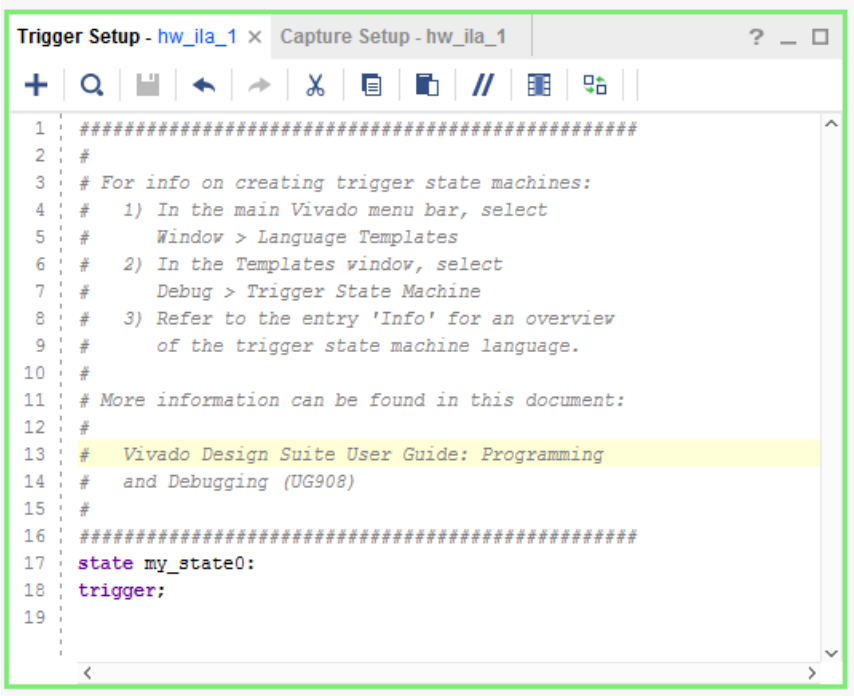


要创建新的触发器状态机，请单击“Create new trigger state machine”（创建新的触发器状态机）链接，否则请单击“Open existing trigger state machine”（打开现有触发器状态机）链接以打开触发器状态机程序文件（扩展名为 `.tsm`）。您还可以使用“Trigger state machine”（触发器状态机）文本字段和/或 ILA 仪表板的“ILA Properties”窗口中的“Browse”（浏览）按钮来打开现有触发器状态机程序文件。

编辑触发器状态机程序

如果您创建了新的触发器状态机程序文件，那么默认情况下会以简单触发器状态机来填充“Trigger State Machine Code Editor”（触发器状态机代码编辑器）窗口（请参阅下图）。

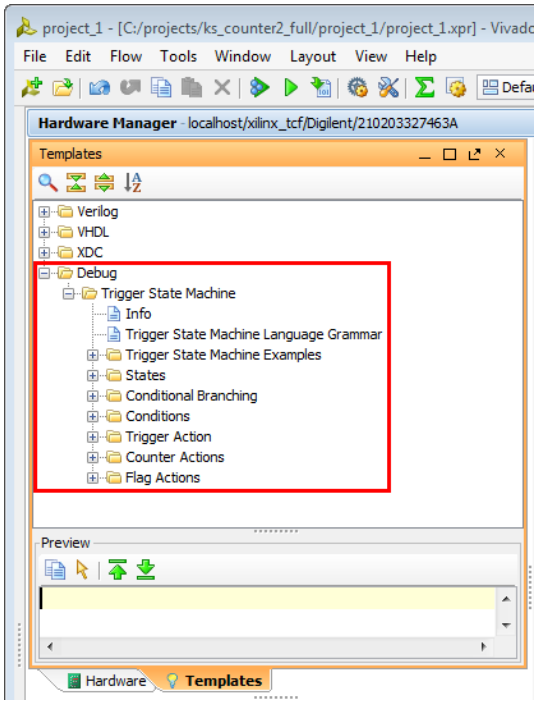
图 120：**简单默认触发器状态机程序**



简单默认触发器状态机程序设计为对于任何 ILA 核配置都有效，与调试探针或触发器设置无关。这意味着您可针对 ILA 核单击“Run Trigger”（运行触发器），而无需修改触发器状态机程序。

但您可修改触发器状态机程序以实现某些高级触发条件。位于上图所示简单状态机顶部的注释块提供了相关信息，以指示如何在 Vivado IDE 中使用内置语言模板来构造触发器状态机程序（请参阅下图）。在“触发器状态机语言描述”中提供了 ILA 触发器状态机语言的完整描述（包括示例）。

图 121：**触发器状态机语言模板**



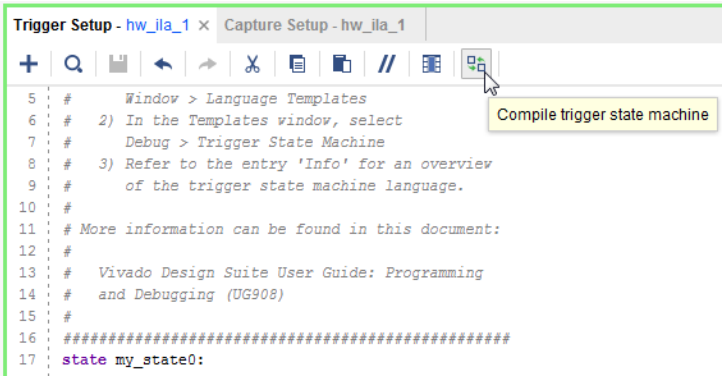
相关信息

触发器状态机语言描述

编译触发器状态机

每次运行 ILA 触发器时，都会编译触发器状态机。如果您想编译触发器状态机而不运行或装备 ILA 触发器，请单击“ILA Dashboard”（ILA 仪表板）中的“Compile trigger state machine”（编译触发器状态机）按钮（如下图所示）。

图 122：**在不装备触发器的前提下编译触发器状态机**



启用触发器输入和输出端口

ILA 核可在核生成时配置为包含专用触发器输入端口（TRIG_IN 和 TRIG_IN_ACK）和专用触发器输出端口（TRIG_OUT 和 TRIG_OUT_ACK）。如果 ILA 核启用触发器输入端口，那么您可使用以下“Trigger Mode”（触发器模式）设置在 TRIG_IN 端口上触发事件：

- BASIC_OR_TRIG_IN：用于在 ILA 核的 TRIG_IN 管脚执行逻辑 OR 操作并且目标模式为 BASIC_ONLY 触发器模式时触发 ILA 核。
- ADVANCED_OR_TRIG_IN：用于在 ILA 核的 TRIG_IN 管脚执行逻辑 OR 操作并且目标模式为 ADVANCED_ONLY 触发器模式时触发 ILA 核。
- TRIG_IN_ONLY：用于在 ILA 核的 TRIG_IN 管脚从低位转换至高位时触发 ILA 核。

如果 ILA 核启用触发器输出端口，那么您可使用以下 TRIG_OUT 模式来控制将触发事件传输至 TRIG_OUT 端口的过程：

- DISABLED：禁用 TRIG_OUT 端口。
- TRIGGER_ONLY：启用将基本/高级触发器条件的结果传输至 TRIG_OUT 端口的过程。
- TRIG_IN_ONLY：将 TRIG_IN 端口传输至 TRIG_OUT 端口。
- TRIGGER_OR_TRIG_IN：启用将基本/高级触发器条件的逻辑 OR 操作结果和 TRIG_IN 端口传输至 TRIG_OUT 端口的过程。

配置捕获模式设置

当 ILA 核状态为“Pre-Trigger”（触发前）、“Waiting for Trigger”（等待触发）或“Post-Trigger”（触发后）时，即可捕获数据样本，请参阅“查看触发和捕获设置”章节了解更多详细信息。“Capture”（捕获）模式控制用于选择捕获每个样本前的条件：

- “ALWAYS”（始终）：在给定时钟周期内存储数据样本，忽略任何捕获条件
- “BASIC”（基本）：仅当捕获条件求值结果为“true”时，才在给定时钟周期内存储数据样本

您也可以使用 `set_property` Tcl 命令来更改 ILA 核的捕获模式。例如，要将 ILA 核 `hw_ila_1` 的捕获模式更改为“BASIC”，请使用以下命令：

```
set_property CONTROL.CAPTURE_MODE BASIC [get_hw_ilas hw_ila_1]
```

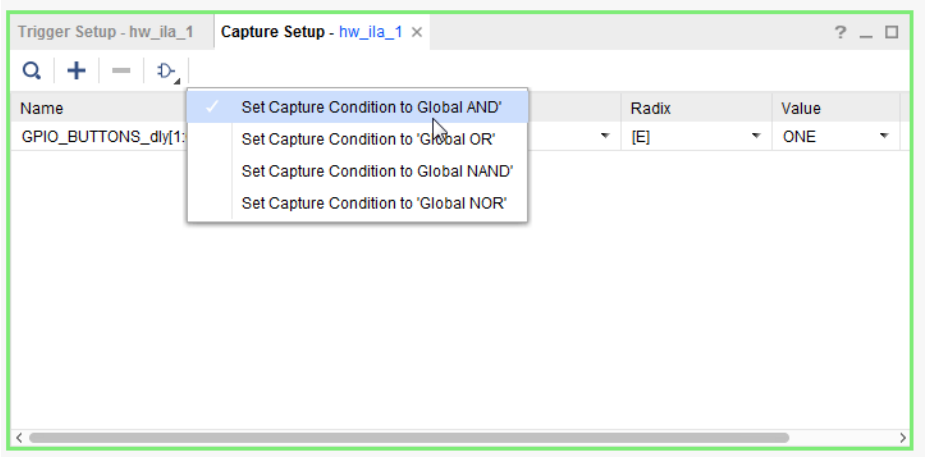
相关信息

查看触发和捕获状态

使用基本捕获模式

使用基本捕获模式来描述捕获条件，即由参与其中的调试探针比较器组成的全局布尔公式。使用“Basic Capture Setup”（基本捕获设置）窗口（请参阅下图）来创建此捕获条件和调试探针比较值。

图 123：**设置基本捕获条件**



配置“基本捕获设置”窗口

在“Basic Capture Setup”（基本捕获设置）窗口中配置调试探针和基本捕获条件的进程与在“Basic Trigger Setup”（基本触发器设置）窗口中处理调试探针的进程非常类似：

- 如需了解有关在“Basic Capture Setup”窗口中添加探针的信息，请参阅以下章节：在“Basic Trigger Setup”窗口中添加探针。
- 如需了解有关在“Basic Capture Setup”窗口中的每个探针上设置比较值的信息，请参阅“ILA 探针比较值设置”部分。
- 如需了解有关在“Basic Capture Setup”窗口中设置基本捕获条件的信息，请参阅“设置基本触发条件”部分。主要差异之一在于用于控制捕获条件的 ILA 核属性称为 CONTROL.CAPTURE_CONDITION。

相关信息

在“Basic Trigger Setup”窗口中添加探针
ILA 探针比较值设置
设置基本触发条件

设置捕获窗口数量

ILA 捕获数据缓冲器可拆分为 1 个或多个捕获窗口。每个窗口的深度均为 2 次幂，它表示样本数，范围为 1 到 (((缓冲器大小) / (窗口数)) - 1)。例如，如果 ILA 数据缓冲器深度为 1024 个样本，且分为 4 个捕获窗口，那么每个窗口深度最大为 256 个样本。每个捕获窗口都有各自独立的触发器掩码，对应于触发器事件，此类事件会导致填充捕获窗口。

注释：不支持每个 ILA 缓冲器大小等于捕获窗口数。

提示：请在整个 ILA 数据捕获缓冲器填满之前单击“Stop Trigger”（停止触发器），以上传并显示填充的所有捕获窗口。例如，如果 ILA 数据缓冲器分为 4 个窗口，其中 3 个已填充数据，那么单击“Stop Trigger”就会停止 ILA 核，然后上传并显示 3 个已填充的捕获窗口。

下表显示了将“Number of Capture Windows”（捕获窗口数）设为大于 1 的值并将“Trigger Position”（触发器位置）设为 0 时，Vivado 运行时软件与硬件之间的交互。

表 10：**捕获窗口数 > 1 且触发器位置 = 0**

软件	硬件
在 ILA 核上使用“Runs Trigger”（运行触发器）	窗口 0：装备 ILA 窗口 0：触发 ILA 窗口 0：ILA 填充捕获窗口 0 窗口 1：重新装备 ILA 窗口 1：触发 ILA 窗口 1：ILA 填充捕获窗口 1 ... 窗口 n-1：重新装备 ILA 窗口 n-1：触发 ILA 窗口 n-1：ILA 填充捕获窗口 n 整个 ILA 捕获缓冲器填满
上传并显示数据	ILA 核会重新装备并准备就绪，以便在上一个窗口的最后一个样本完成捕获后的时钟周期内立即触发。

下表显示了将“Number of Capture Windows”设为大于 1 的值并将“Trigger Position”设为大于 0 的值时，Vivado 运行时软件与硬件之间的交互。

表 11：**捕获窗口数 > 1 且触发器位置 > 0**

软件	硬件
在 ILA 核上使用“Runs Trigger”（运行触发器）	窗口 0：装备 ILA 窗口 0：ILA 填充捕获缓冲器，直至触发器位置为止 窗口 0：触发 ILA 窗口 0：ILA 填充捕获窗口 0 的其余部分 窗口 1：重新装备 ILA 窗口 1：ILA 填充捕获缓冲器，直至触发器位置为止 窗口 1：触发 ILA 窗口 1：ILA 填充捕获缓冲器 窗口 1：ILA 填充窗口 1 ... 窗口 n-1：重新装备 ILA 窗口 n-1：ILA 填充捕获缓冲器，直至触发器位置为止 窗口 n-1：触发 ILA 窗口 n-1：ILA 填充捕获缓冲器 窗口 n-1：ILA 填充窗口 n 整个 ILA 捕获缓冲器填满
上传并显示数据	在 2 个相邻窗口间可能错过触发器，因为 ILA 现在必须填充捕获数据直至触发器位置为止。

在捕获窗口中设置触发器位置

“Capture Mode Settings”（捕获模式设置）窗口中的“Trigger position control”（触发器位置控制）或“ILA Core Properties”（ILA 核属性）窗口中的“Trigger Position”（触发器位置）属性可用于设置捕获的数据窗口中的触发器标记的位置。您可将触发器位置设置为捕获的数据窗口内的任意样本编号。例如，如果捕获的数据窗口深度为 1024 个样本：

- 样本编号 0 对应于捕获的数据窗口中的第一个（最左侧）样本。
- 样本编号 1023 对应于捕获的数据窗口中的最后一个（最右侧）样本。

- 样本编号 511 和 512 对应于捕获的数据窗口中的 2 个“中心”样本。

您也可以使用 `set_property` Tcl 命令来更改 ILA 核触发器位置：

```
set_property CONTROL.TRIGGER_POSITION 512 [get_hw_ilas hw_ila_1]
```

设置捕获窗口的数据深度

“Capture Mode Settings”（捕获模式设置）窗口中的“Data Depth control”（数据深度控制）或“ILA Core Properties”（ILA 核属性）窗口中的“Capture data depth”（捕获数据深度）属性可用于设置 ILA 核捕获的数据窗口的数据深度。您可将数据深度设置为从 1 到 ILA 核的最大数据深度之间的任意 2 次幂值，该值在核生成或插入时间之间设置。

注释：请参阅 设置捕获窗口的数据深度，以详细了解在使用“网表插入”探测流程添加到设计中的 ILA 核上，如何设置最大捕获缓冲器数据深度。

您也可以使用 `set_property` Tcl 命令来更改 ILA 核数据深度：

```
set_property CONTROL.DATA_DEPTH 512 [get_hw_ilas hw_ila_1]
```

相关信息

在调试核上修改属性

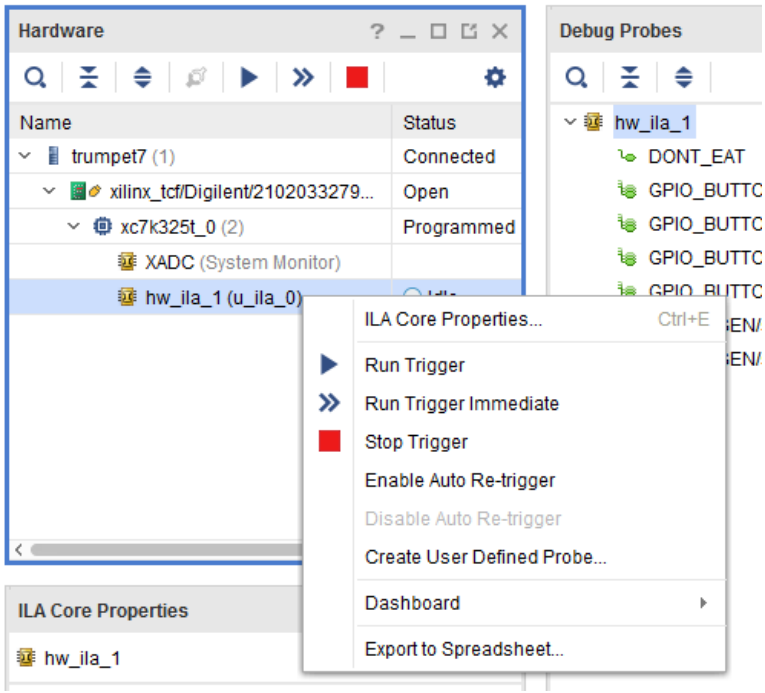
运行触发器

您可在 2 种不同模式下运行或装备 ILA 核触发器：

- “Run Trigger”（运行触发器）：选择要装备的 ILA 核，然后单击“ILA Dashboard”（ILA 仪表板）窗口或“Hardware”（硬件）窗口工具栏上的“Run Trigger”按钮即可装备 ILA 以检测 ILA 核的基本触发器或高级触发器设置所定义的触发事件。
- “Run Trigger Immediate”（立即运行触发器）：选择要装备的 ILA 核，然后单击“ILA Dashboard”或“Hardware”硬件窗口工具栏上的“Run Trigger Immediate”按钮即可装备 ILA 核以便立即触发（忽略 ILA 核触发器设置）。此命令用于通过捕获 ILA 核的探针输入处的任意活动来检测设计的“活动状态”。

您还可通过选中并右键单击 ILA 核，然后从弹出菜单中单击“Run Trigger”（运行触发器）或“Run Trigger Immediate”（立即运行触发器）来装备触发器（请参阅下图）。

图 124：**ILA 核触发器命令**



提示：您可通过选中目标 ILA 核，然后使用“Hardware”（硬件）窗口工具栏中的“Run Trigger”（运行触发器）、“Run Trigger Immediate”（立即运行触发器）或“Stop Trigger”（停止触发器）按钮来运行或停止多个 ILA 核的触发器。您也可以通过在“Hardware”窗口中选中给定器件并单击“Hardware”窗口工具栏中的相应按钮来运行或停止触发该器件中的所有 ILA 核。

停止触发器

您可通过选中相应的 ILA 核，然后单击 ILA 仪表板或“Hardware”（硬件）窗口工具栏上的“Stop Trigger”（停止触发器）按钮来停止 ILA 核触发器。也可选中并右键单击相应的 ILA 核，然后从弹出菜单中单击“Stop Trigger”来停止触发器（请参阅“运行触发器”）。

相关信息

运行触发器

使用自动重新触发

选择 ILA 核上的“Enable Auto Re-Trigger”（启用自动重新触发）右键菜单选项（或 ILA 仪表板工具栏上的对应按钮），这样在成功完成触发 + 上传 + 显示操作后，即可启用 Vivado IDE 以自动重新装备 ILA 核触发器。每次成功完成触发事件后，对应于 ILA 核的波形查看器中显示的捕获数据都会被覆盖。“Auto Re-Trigger”（自动重新触发）选项可搭配“Run Trigger”（运行触发器）操作和“Run Trigger Immediate”（立即运行触发器）操作一起使用。单击“Stop Trigger”（停止触发器）即可停止当前运行中的触发器。

下表显示了调用“Auto Re-Trigger”（自动重新触发）选项时 Vivado IDE 运行时软件与硬件之间的交互。

表 12：**自动重新触发**

软件	硬件
单击 ILA 核上的“Auto Re-trigger”选项	ILA 完成装备 ILA 触发 ILA 填充捕获缓冲器 ILA 已满
上传并显示数据	ILA 重新装备 ILA 触发 ILA 填充捕获缓冲器 ILA 已满 在 ILA“已满”事件与显示 ILA 数据之间有大量周期丢失

重要提示！由于 ILA 数据装满与在 GUI 中上传并显示数据之间存在延迟，因此在这些事件之间丢失周期的可能性极高，而在此期间可能触发 ILA。

查看触发和捕获状态

在 Vivado IDE 中，ILA 调试核触发和捕获状态显示在以下 2 处位置：

- 在对应于各 ILA 调试核的各个行的“Hardware”（硬件）窗口“Status”（状态）列中。
- 在 ILA 仪表板的“Trigger Capture Status”（触发捕获状态）窗口中。

“Hardware”窗口的“Status”列可指示每个 ILA 核的当前状态（请参阅下表）。

表 13：**ILA 核状态描述**

ILA 核状态	描述
Idle	ILA 核处于空闲状态，正在等待其触发器运行。如果触发器位置为 0，那么一旦触发器运行，ILA 核就会转换为“Waiting for Trigger”（等待触发）状态，否则，ILA 核会转换为“Pre-Trigger”（触发前）状态。
Pre-Trigger	ILA 核正在将触发前数据捕获到其数据捕获窗口内。一旦捕获到触发前数据，ILA 核就会转换为“Waiting for Trigger”状态。
Waiting for Trigger	ILA 核触发器已装备，正在等待发生基本或高级触发器设置中所述的触发器事件。一旦发生触发事件，如果触发器位置设为捕获窗口中最后一个数据样本的位置，ILA 核就会转换为“Full”（完全）状态，否则它将转换为“Post-Trigger”（触发后）状态。
Post-Trigger	ILA 核正在将触发后数据捕获到其数据捕获窗口内。一旦捕获到触发后数据，ILA 核就会转换为“Full”（完全）状态。
Full	ILA 核捕获缓冲器已满，并且正在上传到主机以供显示。一旦上传并显示了数据，ILA 核就会转换到“Idle”（空闲）状态。

ILA 仪表板中“Trigger Capture Status”窗口的内容取决于 ILA 核的“Trigger Mode”（触发器模式）设置。

部分缓冲器捕获

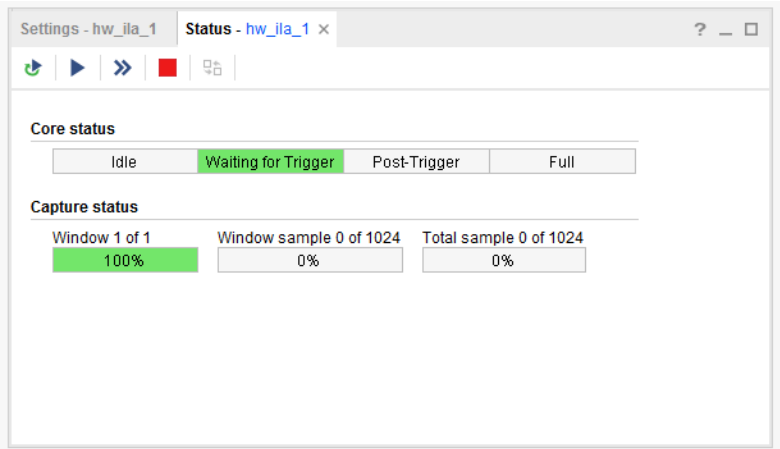
在整个 ILA 数据捕获缓冲器填满之前单击“Stop Trigger”（停止触发器）就会上传并显示所有捕获窗口。例如，如果 ILA 数据缓冲器分成 4 个窗口，并且其中 3 个窗口已填充数据，那么单击“Stop Trigger”就会停止 ILA 核，然后上传并显示 3 个已填充的捕获窗口。此外，单击“Stop Trigger”还会停止 ILA 核，并显示已部分填充的捕获窗口，前提是在此捕获窗口内发生了触发事件。

基本触发器模式下的触发和捕获状态

当“Trigger Mode”（触发器模式）设置为“BASIC”（基本）模式时，“Trigger Capture Status”（触发捕获状态）窗口包含 2 个状态指示器（请参阅下图）：

- “Core status”（核状态）：指示 ILA 核触发/捕获引擎的状态（请参阅“查看触发和捕获状态”以获取状态指示器的描述）。
- “Capture status”（捕获状态）：指示当前捕获窗口、当前捕获窗口中捕获的当前样本数以及 ILA 核捕获的样本总数。当 ILA 核状态为“Idle”（空闲）时，这些值全部复位为 0。

图 125：**基本触发器模式下的触发捕获状态窗口**



相关信息

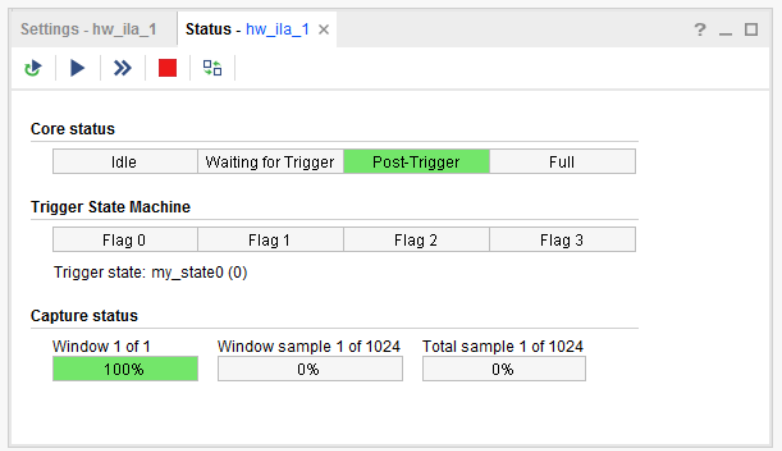
查看触发和捕获状态

高级触发器模式下的触发和捕获状态

当“Trigger Mode”（触发器模式）设置为“ADVANCED”（高级）时，“Trigger Capture Status”（触发器捕获状态）窗口包含 4 个状态指示器（请参阅下图）：

- “Core status”（核状态）：指示 ILA 核触发/捕获引擎的状态（请参阅“查看触发和捕获状态”以获取状态指示器的描述）。
- “Trigger State Machine Flags”（触发器状态机标志）：指示 4 个触发器状态机标志的当前状态。
- “Trigger State”（触发器状态）：当核状态为“Waiting for Trigger”（等待触发）时，该字段指示触发器状态机的当前状态。
- “Capture status”（捕获状态）：指示当前捕获窗口、当前捕获窗口中捕获的当前样本数以及 ILA 核捕获的样本总数。当 ILA 核状态为“Idle”（空闲）时，这些值全部复位为 0。

图 126：**高级触发器模式下的“Trigger Capture Status”窗口**



相关信息

查看触发和捕获状态

写入 ILA 探针信息

“Debug Probes”（调试探针）窗口中的“ILA Cores”（ILA 核）选项卡视图包含有关您在自己的设计中使用 ILA 核探测的信号线的信息。此 ILA 探针信息提取自您的设计，并存储在数据文件内，此数据文件通常带有 `.ltx` 文件扩展名。

通常，ILA 探针文件是在比特流生成期间自动创建的。但是，您也可以使用 `write_debug_probes` Tcl 命令来将调试探针信息写出至文件：

1. 如果使用工程模式，请打开“Implemented Design”（已实现的设计）。如果使用非工程模式，请打开已实现的设计检查表。
2. 运行 `write_debug_probes` 文件名 `.ltx` Tcl 命令。

读取 ILA 探针信息

如果在与器件关联的比特流烧录 (`.bit`) 文件所在目录内找到了名为 `debug_nets.ltx` 的 ILA 探针文件，那么此探针文件会与 FPGA 或自适应 SoC 硬件器件自动关联。

您还可以指定探针文件的位置：

1. 在“Hardware”（硬件）窗口内选择 FPGA 或自适应 SoC。
2. 在“Hardware Device Properties”（硬件器件属性）窗口中设置探针文件位置。
3. 单击“Apply”（应用）以应用更改。

您也可以使用 `set_property` Tcl 命令来设置位置：

```
set_property PROBES.FILE {C:/myprobes.ltx} [lindex [get_hw_devices] 0]
```

在波形查看器中查看从 ILA 核捕获的数据

当 ILA 核捕获的数据上传到 Vivado IDE 后，就会显示在“Waveform Viewer”（波形查看器）中。请参阅“查看 ILA 探测数据”以获取有关使用波形查看器来查看从 ILA 核捕获的数据的详细信息。

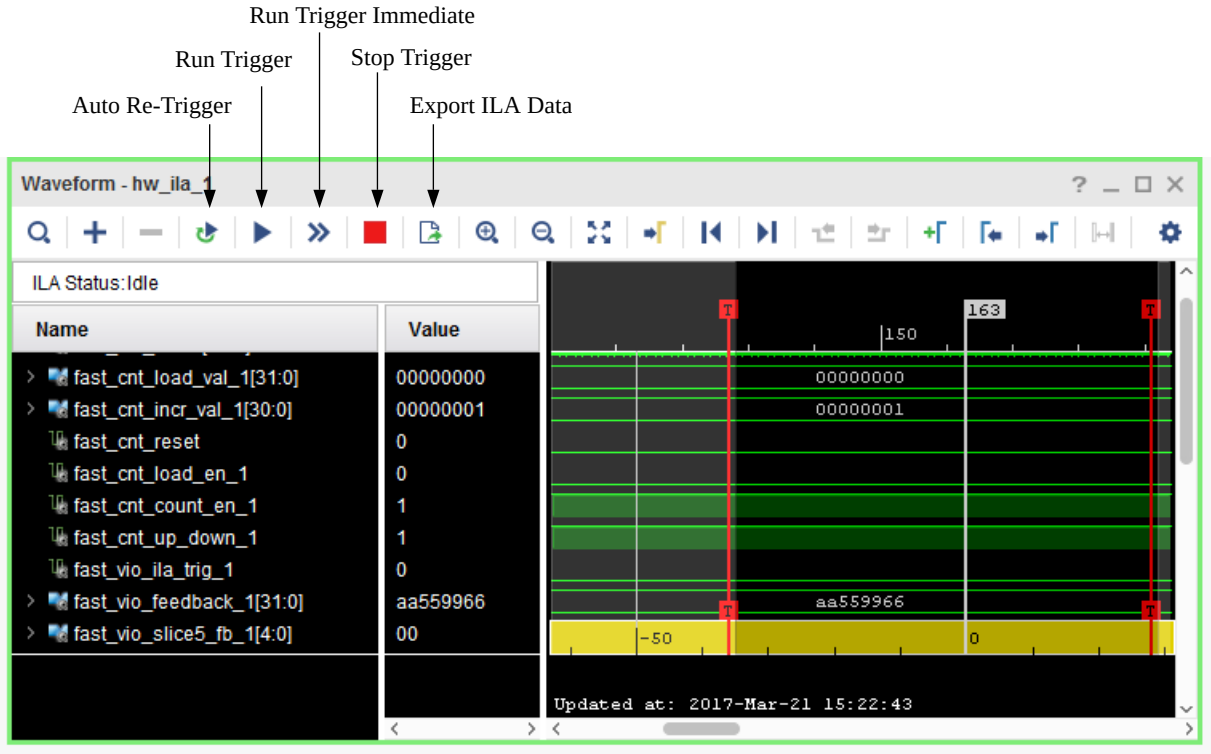
相关信息

在波形查看器中查看 ILA 探针数据

使用波形 ILA 触发器和导出功能

您可以使用“Waveform”（波形）窗口中的图标来装备 ILA 和运行触发器、停止触发器以及导出 ILA 数据，如下所示：

图 127：**波形 ILA 触发器和导出功能**



X16755-032717

“Enable Auto Re-Trigger”（启用自动重新触发）：选中“Waveform”（波形）窗口工具栏上的“Enable Auto Re-Trigger”按钮即可在成功完成触发 + 上传 + 显示操作后，启用 Vivado IDE 以自动重新装备与“Waveform”窗口触发器关联的 ILA 核。

每次成功完成触发事件后，对应于 ILA 核的“Waveform”窗口中显示的捕获数据都会被覆盖。“Auto Re-Trigger”（自动重新触发）选项可搭配“Run Trigger”（运行触发器）操作和“Run Trigger Immediate”（立即运行触发器）操作一起使用。单击“Stop Trigger”（停止触发器）按钮即可停止当前运行中的触发器。

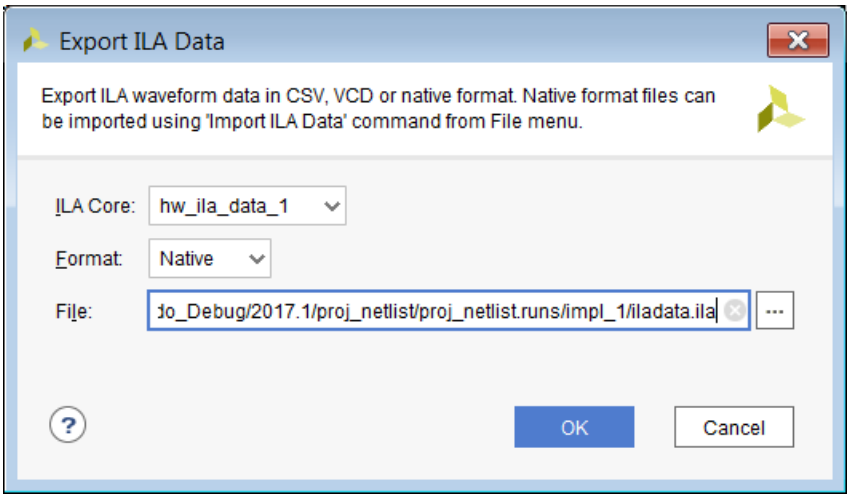
“Run Trigger”：装备与“Waveform”窗口关联的 ILA 核，以检测由 ILA 核的基本或高级触发器设置所定义的触发器事件。

“Run Trigger Immediate”：装备与“Waveform”窗口关联的 ILA 核以忽略 ILA 核触发器设置，并立即触发该核。此命令用于通过捕获 ILA 核的探针输入处的任意活动来检测设计的“活动状态”。

“Stop Trigger”：停止与“Waveform”窗口关联的 ILA 的 ILA 核触发器。

“Export ILA Data”（导出 ILA 数据）：捕获来自 ILA 核的数据并将其保存到文件。此数据可采用本机格式、`.csv` 或 `.vcd` 格式来捕获。在“Waveform”（波形）窗口工具栏上单击此图标后，会显示以下对话框。

图 128：**“Export ILA Data”对话框**



“ILA Core”（ILA 核）表示要为其导出数据的 ILA 调试核的名称。“Format”（格式）支持下列格式：Native（本机格式）、CSV 和 VCD。

- 本机格式可配置 `write_hw_ila_data` 命令，按本机 ILA 格式来导出 ILA 数据。

  此 ILA 文件可重新导入 Vivado IDE 以便您查看先前捕获的 ILA 数据。`read_hw_ila_data` Tcl 命令可用于将 ILA 数据重新导入 Vivado IDE 中，如以下示例所示：

  ```
  read_hw_ila_data {./iladata.ila} display_hw_ila_data
  ```

- CSV 格式可配置 `write_hw_ila_data` 命令，按 `.csv` 文件格式导出 ILA 数据，此文件可用于将数据导入电子表格或第三方应用。
- VCD 文件格式可配置 `write_hw_ila_data` 命令，按 `.vcd` 文件格式导出 ILA 数据，此格式可用于导入第三方应用或查看器。

> **重要提示！**虽然 ILA 数据可按 CSV、VCD 和本机 ILA 格式导出，但在 Vivado 中只能导入本机 ILA 格式。并且，仅支持将本机 ILA 数据导入 Vivado 中用于脱机查看先前捕获的数据。探针信号不能用于其他目的，例如触发等。

保存和复原从 ILA 核捕获的数据

除了显示从 ILA 核直接上传的捕获数据外，您还可以将这些捕获数据写入文件，然后从文件中读取这些数据，并将其显示在波形查看器中。

将捕获的 ILA 数据保存到文件

当前将从 ILA 核捕获的数据上传并保存到文件的唯一方法是使用以下 Tcl 命令：

```
write_hw_ila_data my_hw_ila_data_file.ila [upload_hw_ila_data hw_ila_1]
```

此 Tcl 命令序列会将从 ILA 核捕获的数据上传并写入名为 `my_hw_ila_data_file.ila` 的存档文件。此存档文件包含波形数据库文件、波形配置文件、波形逗号分隔值文件以及调试探针文件。

提示：使用 `-csv_file` 选项可生成逗号分隔值 (CSV) 文件。这样即可配置 `write_hw_ila_data` 命令以将 ILA 数据导出至 CSV 格式（而不是默认二进制 ILA 文件格式）的文件，此文件可供用于导入电子表格或第三方应用。

提示：使用 `-vcd_file` 选项可生成值变转储 (VCD) 文件。这样即可配置 `write_hw_ila_data` 命令以将 ILA 数据导出至 VCD 格式（而不是默认二进制 ILA 文件格式）的文件，此文件可用于导入第三方应用或查看器。

探针数据基数

每个探针都具有关联的 DISPLAY_RADIX 属性。默认情况下，该属性针对多位探针设置为 HEX，针对单位探针则设置为 BINARY。`.csv` 文件中导出的探针数据会使用探针基数。

您可以在 Vivado 硬件管理器 Tcl 控制台中更改设计中所有 ILA 的所有探针的 DISPLAY_RADIX 属性，如下所示：

```
foreach probe [get_hw_probes -of [get_hw_ilas]] {   set_property
DISPLAY_RADIX binary $probe   set_property DISPLAY_AS_ENUM false $probe  }
```

注释：此处您所做的是将所有 ILA 中的所有探针基数更改为 BINARY。要将基数更改为 HEX，请使用以下脚本：

```
foreach probe [get_hw_probes -of [get_hw_ilas]] {   set_property
DISPLAY_RADIX hex $probe   set_property DISPLAY_AS_ENUM false $probe  }
```

与单一探针关联的数据样本列表

您还可使用 `list_hw_samples` Tcl 命令来列示与各探针关联的数据样本。

以下示例列出了与名为 `i_fast_ila` 的 ILA 上的 `fast_cnt_incr_val_1` 探针关联的样本：

```
list_hw_samples [get_hw_probes fast_cnt_incr_val_1 -of_objects [get_hw_ilas
-of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001...
```

从文件恢复捕获的 ILA 数据

当前，从文件复原捕获的数据并将其显示在波形查看器中的唯一方法是使用以下 Tcl 命令：

```
display_hw_ila_data [read_hw_ila_data my_hw_ila_data_file.ila]
```

此 Tcl 命令序列会读取先前从 ILA 核捕获并保存的数据，然后将其显示在“Waveform”（波形）窗口中。

注释： “ILA Data Waveform”（ILA 数据波形）窗口的波形配置设置（分频器、标记、颜色、探针基数等）也同样保存在 ILA 捕获数据存档文件中。恢复和显示先前保存的任意 ILA 数据时，会使用这些存储的波形配置设置。

重要提示！ 请勿使用 `open_wave_config` 命令打开根据 ILA 捕获的数据创建的波形。这条仅适用于仿真器的命令对于 ILA 捕获的数据波形无效。

探针值枚举

此功能支持按符号名称引用探针值，既可在“Trigger/Capture Setup”（触发/捕获设置）期间引用探针值进行比较，也可在“Waveform”（波形）窗口中引用探针值。

以下提供了常见用例。

- 状态机状态值
- 操作代码
- 任意探针匹配值，以供您按名称（而不是按值）来引用。

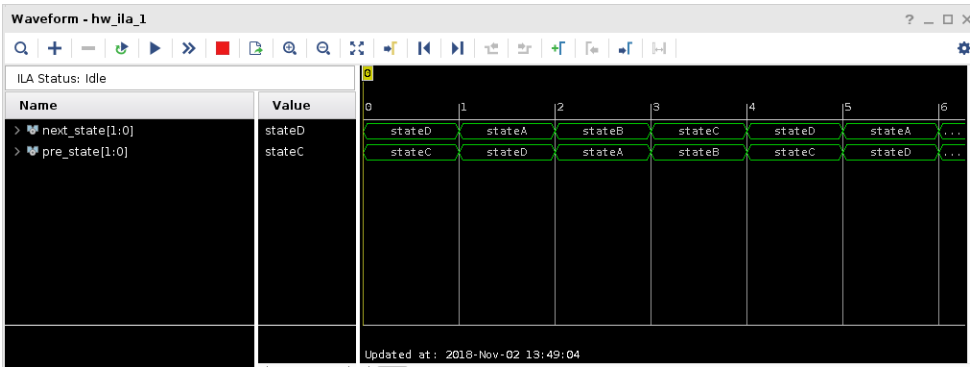
此进程包括输入枚举名称/值对，以及将其与探针关联。枚举名称/值对关联可使用 Tcl 命令来执行并在会话间进行存储。

如果使用 mark_debug 属性来探测状态机中含枚举状态的状态寄存器，那么可通过实现来保留状态枚举，并自动显示在“Waveform”窗口中。

图 129：**含状态编码的 RTL**

```
14
15  (* mark_debug = "true" *) reg [1:0] pre_state;
16  (* mark_debug = "true" *) reg [1:0] next_state;
17
18  localparam stateA = 2'b00;
19  localparam stateB = 2'b01;
20  localparam stateC = 2'b10;
21  localparam stateD = 2'b11;
22
```

图 130：**波形查看器中保留并显示为探针枚举的状态编码**



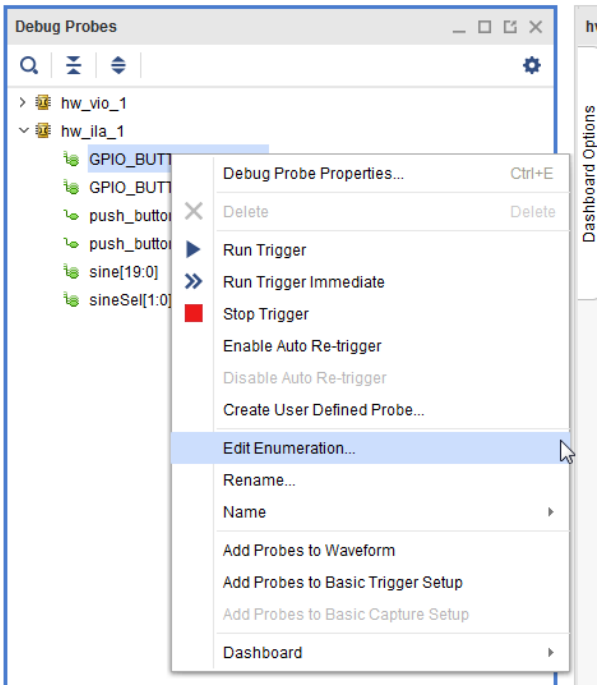
探针比较值可使用“Trigger Setup”（触发器设置）窗口和“Capture Control Setup”（捕获控制设置）窗口中的枚举名称来设置。探针及其枚举名称（对应于值）同样可显示在“Waveform”窗口中。

添加/编辑枚举

使用硬件管理器定义新的枚举

要将新的枚举名称/值对与调试探针相关联，请在“Debug Probes”（调试探针）窗口或“Trigger/Capture Setup”（触发/捕获设置）窗口中右键单击调试探针，然后选中“Edit Enumeration”（编辑枚举）。

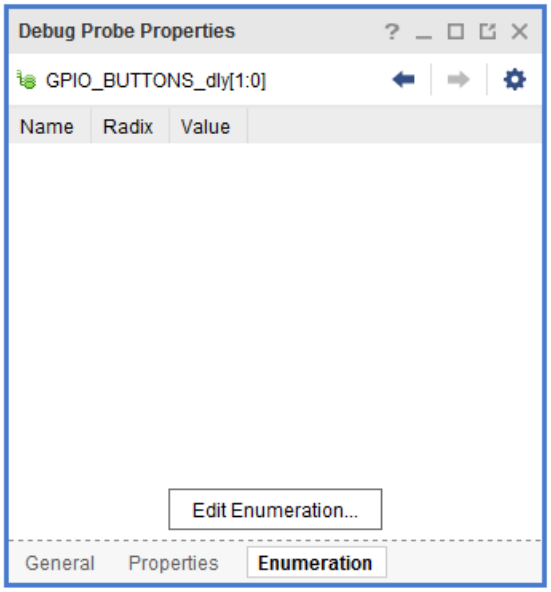
图 131：**在“Trigger Setup”窗口中编辑枚举**



在“Trigger Setup”窗口中编辑与调试探针关联的枚举

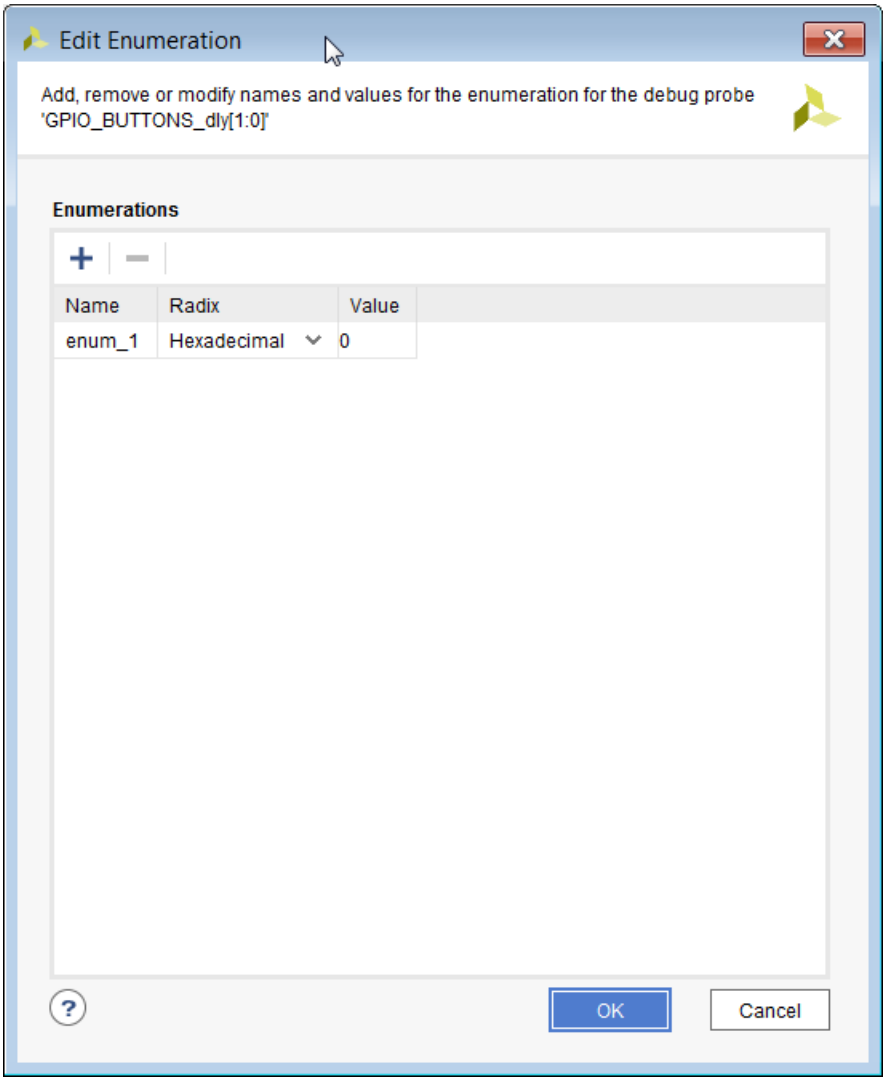
您也可以在“Debug Probes”（调试探针）窗口或“Trigger Setup”（触发器设置）窗口中选中调试探针以将新的枚举名称/值对关联到调试探针。“Debug Probe Properties”（调试探针属性）窗口的“Enumeration”（枚举）选项卡也支持您将新的名称/值对关联到探针。

图 132：**“Debug Probe Properties”对话框**



单击“Edit Enumeration”（编辑枚举）打开“Edit Enumeration”对话框。

图 133：**“Edit Enumeration”对话框**



选择名称/值对，并使用左侧的“+”和“-”按钮来添加或删除枚举。您可更改表中的“Name”（名称）、“Radix”（基数）和“Value”（值）字段。

使用 Tcl 命令添加枚举

`add_hw_probe_enum` 命令可用于将枚举名称/值对与调试探针相关联。您可将 `add_hw_probe` 命令添加到 Tcl 文件中以使其定义显示在独立文件中。枚举名称保留其输入时的大小写，但查找时不区分大小写。

示例：

```
set probe [get_hw_probes fast_cnt_count -of_objects [get_hw_ilas -
of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
add_hw_probe_enum ZERO eq5'h00 $probe add_hw_probe_enum TWELVE eq5'u12
$probe add_hw_probe_enum THIRTEEN eq5'u13 $probe add_hw_probe_enum FOURTEEN
eq5'u14 $probe add_hw_probe_enum FIFTEEN eq5'u15 $probe add_hw_probe_enum
SIXTEEN eq5'u16 $probe add_hw_probe_enum SEVENTEEN eq5'u17 $probe
```

使用 Tcl 命令删除枚举

remove_hw_probe_enum 命令可用于移除明确指定的枚举条目或者 hw_probe 的所有枚举。

示例：

```
remove_hw_probe_enum -list {zero } [get_hw_probes U_SINEGEN/sel -of_objects
[get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter
{CELL_NAME=~"u_ila_0"}]]
```

提示：使用 remove_hw_probe_enum 中的 -remove_all 选项可移除与探针关联的所有枚举。

访问枚举

枚举作为 hw_probe 属性来存储，因此可在这些属性上使用 set_property、get_property 和 report_property 命令。

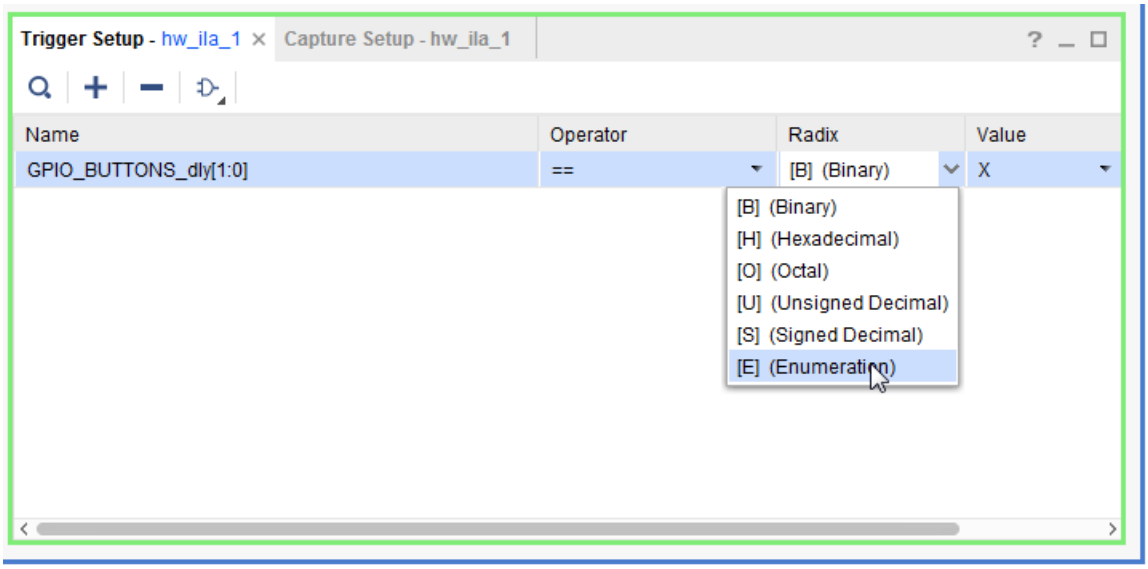
枚举属性带有 ENUM 前缀，将枚举与 set_property 和 get_property 命令配合使用时，需使用此前缀。请参阅以下示例。

```
get_property ENUM.FIFTEEN -of_objects [get_hw_ilas -of_objects
[get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]] eq5'u15
report_property [get_hw_probes fast_vio_slice5_fb -of_objects [get_hw_ilas
hw_ila_1]] ENUM* Property Type Read-only Visible Value ENUM.FIFTEEN string
true true eq5'u15 ENUM.FOURTEEN string true true eq5'u14 ENUM.SEVENTEEN
string true true eq5'u17 ENUM.SIXTEEN string true true eq5'u16
ENUM.THIRTEEN string true true eq5'u13 ENUM.TWELVE string true true eq5'u12
ENUM.ZERO string true true eq5'h00 set_property ENUM.FIFTEEN eq5'h0F
[get_hw_probes fast_vio_slice5_fb -of_objects [get_hw_ilas hw_ila_1]]
```

在“Trigger Setup”窗口中使用枚举

在“Trigger Setup”（触发器设置）窗口中设置比较值，其语法与数字比较值类似。基数字符为“e”。针对枚举比较值，仅支持运算符“eq”和“neq”。

图 134：**“Trigger Setup”窗口中的枚举**




```
set_property TRIGGER_COMPARE_VALUE eq2'ethree [get_hw_probes U_SINEGEN/sel -
of_objects [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter
{CELL_NAME=~"u_ila_0"}]]
```

在“Capture Setup”窗口中使用枚举

您还可在 Vivado IDE 的“Capture Setup”（捕获设置）窗口中使用枚举或者使用 Tcl 命令来比较各值。

图 135：**在“Capture Setup”窗口中使用枚举**


```
set_property CAPTURE_COMPARE_VALUE eq2'eone [get_hw_probes locked -
of_objects [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter
{CELL_NAME=~"u_ila_0_0"}]]
```

高级触发器

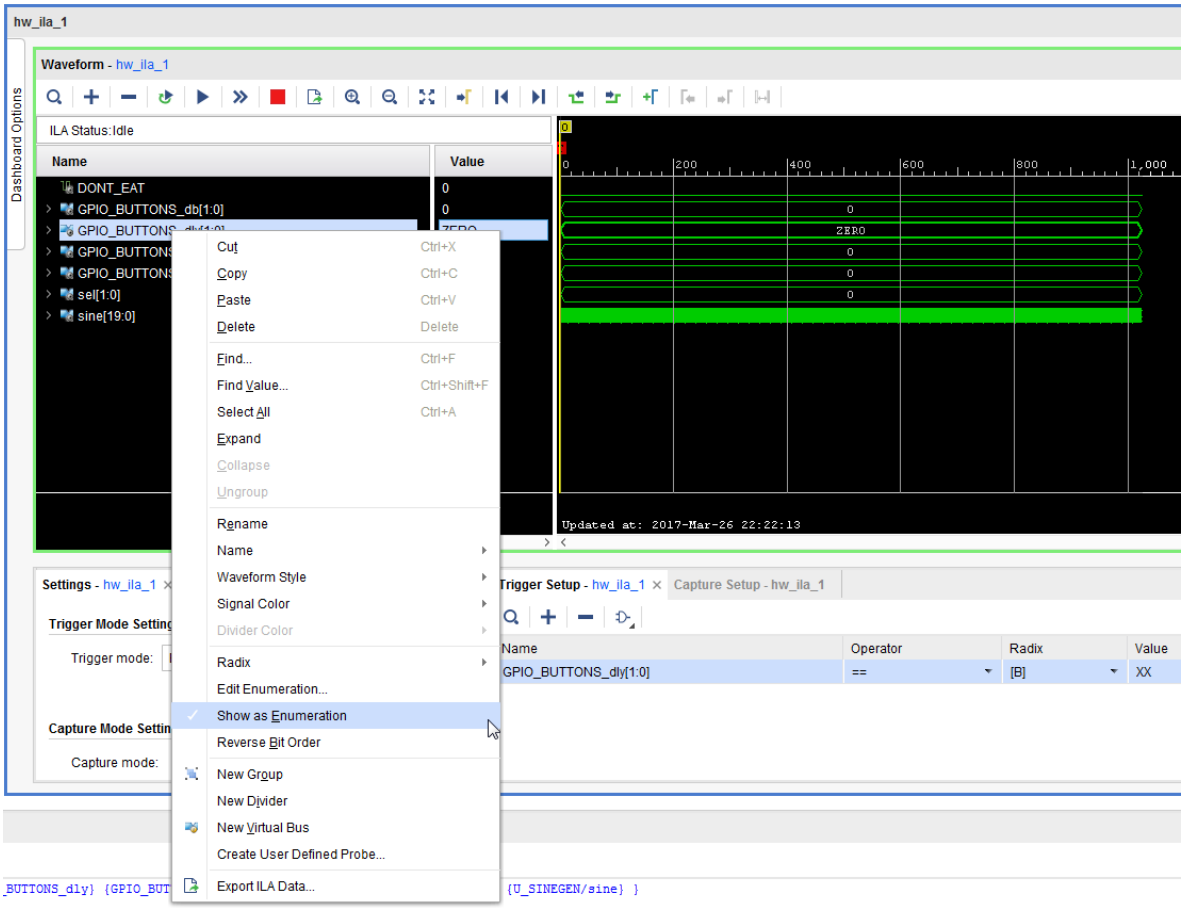
您可在“Advanced Trigger State Machine”（高级触发器状态机）脚本中使用枚举来比较各值，如以下示例所示。

```
state my_state0:  if (fast_ila_slice5_fb == 5'eFIFTEEN) then    set_flag
$flag1;    goto my_state0;  elseif (fast_ila_slice5_fb == 5'eTWELVE) then
   trigger;  else    clear_flag $flag1;    goto my_state0;  endif
```

在“Waveform”窗口中使用枚举

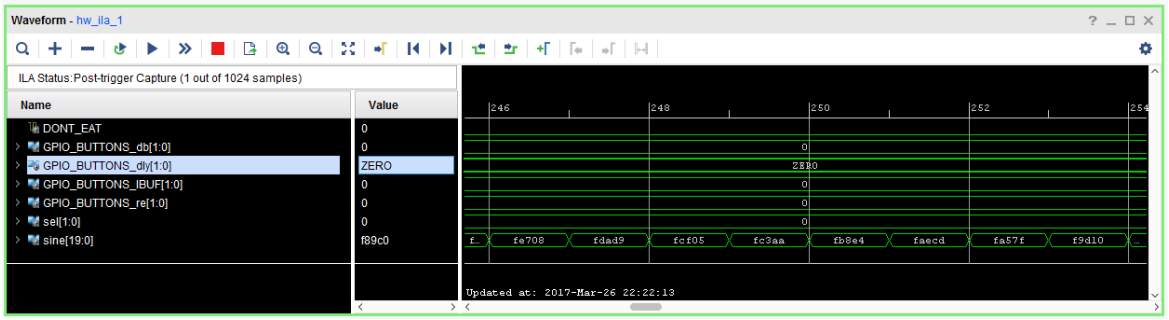
您可在“Waveform”（波形）窗口中针对每个信号选中“Show as Enumeration”（以枚举方式显示）选项来显示枚举。在“Waveform”窗口中右键单击信号，并从显示的菜单中选择“Show as Enumeration”。不以枚举方式显示时，总线值根据常规基数选择来显示。

图 136：**“Waveform”窗口中的“Show as Enumeration”选项**



枚举信息将保存到波形数据文件中，并在后续显示波形数据时使用。已定义“Enumerations”的波形探针的默认设置是显示“Enumerations”。

图 137：**含枚举的波形**



当波形对象选中“Show as Enumeration”时，将显示枚举名称。如果波形值没有匹配的枚举，则会根据所选基数来显示。

重要提示！如果在创建枚举前已创建波形，则可通过使用 Tcl 命令来保存波形 ILA 数据的方式将新枚举应用于波形，如下所示：

```
write_hw_ila_data -force data_ila_3.ila [upload_hw_ila_data hw_ila_3]
display_hw_ila_data [read_hw_ila_data ./data_ila_3.ila]
```

在硬件管理器中调试 AXI 接口

IP integrator 中的 System ILA IP 支持您在 FPGA 上对设计执行系统内调试。在 Versal 器件上，System ILA 核已被弃用。现在，在含 AXIS 接口的标准 ILA 中支持接口调试。如需监控 IP integrator 块设计中的接口和信号，可使用此功能。

请参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994) 中的“创建块设计以使用开发板流程”章节，了解在块设计中调试接口和/或信号线的步骤。

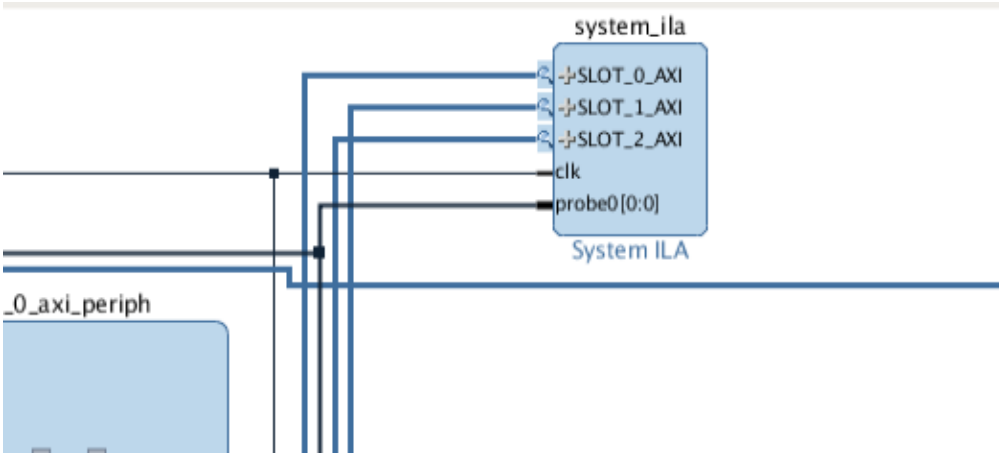
如果在 IP integrator 块设计中已例化 System ILA 调试核，则可在波形窗口中调试并监控 AXI 传输事务及其对应的读写事件。

注释：对于非 Versal 架构，必须使用 System ILA 进行接口调试。

波形和 AXI 接口

System ILA 调试核支持您将接口作为插槽以便进行调试和监控。每个插槽都对应于在 IP integrator 块设计中进行调试的 1 个接口。在下图中显示了 2 个 AXI4 接口，System ILA IP 当前正在插槽 0 和插槽 1 中分别对其进行探测。

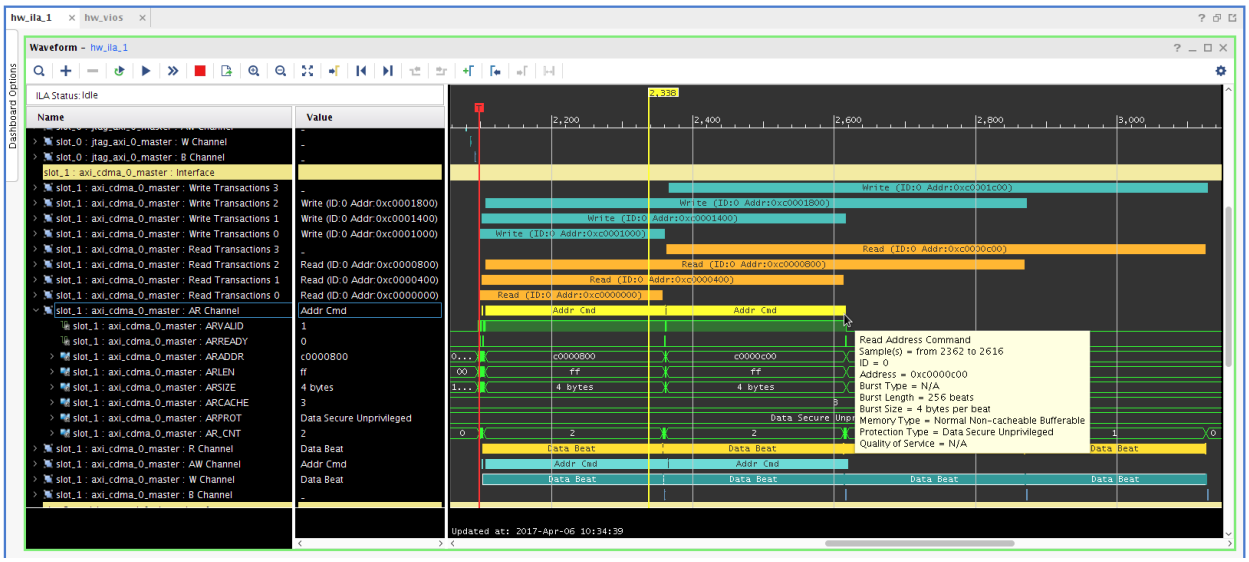
图 138：**探测 2 个 AXI4 接口**



波形查看器中的 AXI 传输事务

在波形查看器中可以查看与 System ILA 所调试的 AXI3、AXI4 和 AXI4-Lite 接口关联的传输事务，如下图所示

图 139：**波形查看器中的 AXI 传输事务**



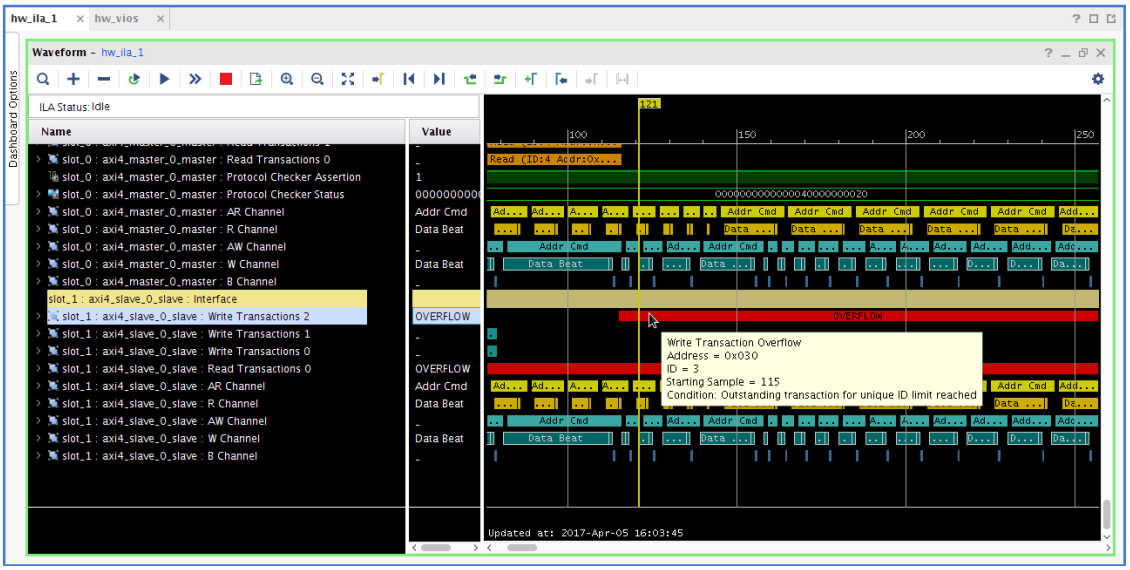
AXI 传输事务定义如下：

- 读取传输事务起始，此类事务随 AR（读取地址）通道上的“Address Command”（地址命令）事件开始而启动。
- 读取传输事务结束，此类事务随 R（读取数据）通道上的“Last Read Data”（最后一次读取数据）事件结束而结束。
- 写入传输事务起始，此类事务随 AW（写入地址）通道上的“Address Command”（地址命令）事件开始而启动。
- 写入传输事务结束，此类事务随 B（写入响应）通道上的“Write Response”（写入响应）事件结束而结束。

仅当地址、数据和/或响应事件具有匹配的 ID 时，才会显示相应的传输事务。此外，仅当捕获的数据波形中起始事件和结束事件都发生时，才会在波形中显示相应的传输事务。当在“Waveform”窗口中显示多个未完成/重叠的传输事务时，会使用多个传输事务行。

接口上的传输事务可能导致 System ILA IP 中未完成的传输事务跟踪逻辑发生上溢，如下图所示。

图 140：**AXI 传输事务计数器上溢状况**

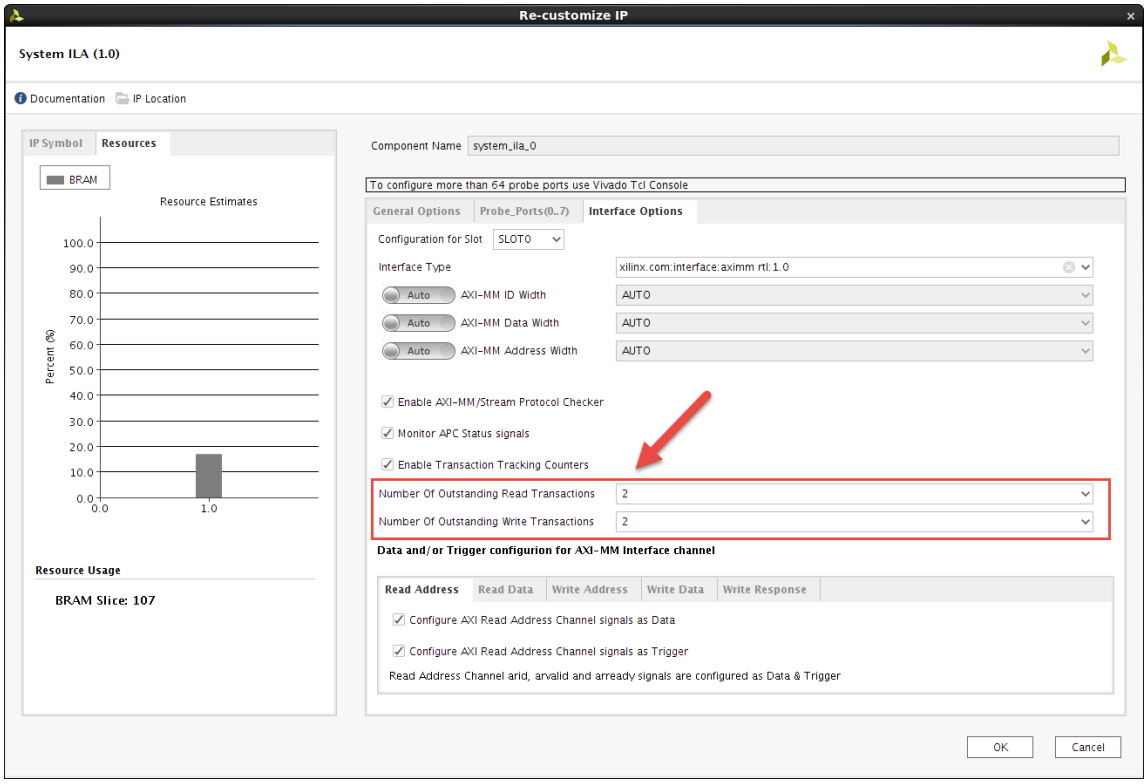


可能出现 2 种上溢状况：

- 特定 ID 的未完成传输事务数量导致传输事务计数器容量上溢。
- 包含未完成的传输事务的 ID 数量导致可用计数器数量上溢。

在上述任一情况下，上溢状况均可通过在 IP integrator 块设计中重新自定义 System ILA 核以增大未完成的读取和/或写入传输事务数量来解决。请参阅下图。

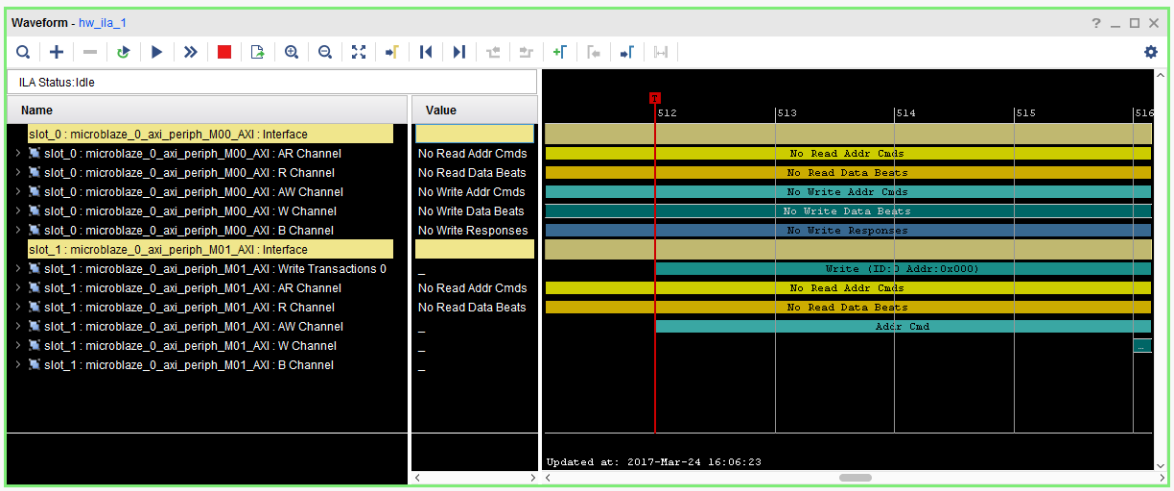
图 141：**增大可供 System ILA 跟踪的未完成的传输事务数量**



AXI 接口事件

在 Vivado 硬件管理器中，如果使用 System ILA IP 对设计 AXI 接口进行调试，那么“Waveform”（波形）窗口会显示对应于 System ILA 所探测的接口的接口插槽、事件和信号组。正如下图所示，“Waveform”（波形）窗口会显示 System ILA IP 所探测的 2 个接口插槽。您可在插槽 1 上查看“AXI 传输事务”、“写入地址通道事件”和“写入数据通道事件”。您还可在“Waveform”窗口中查看“Write Data CAXI”（写入数据 CAXI）接口插槽。

图 142：**AXI 接口传输事务和事件**



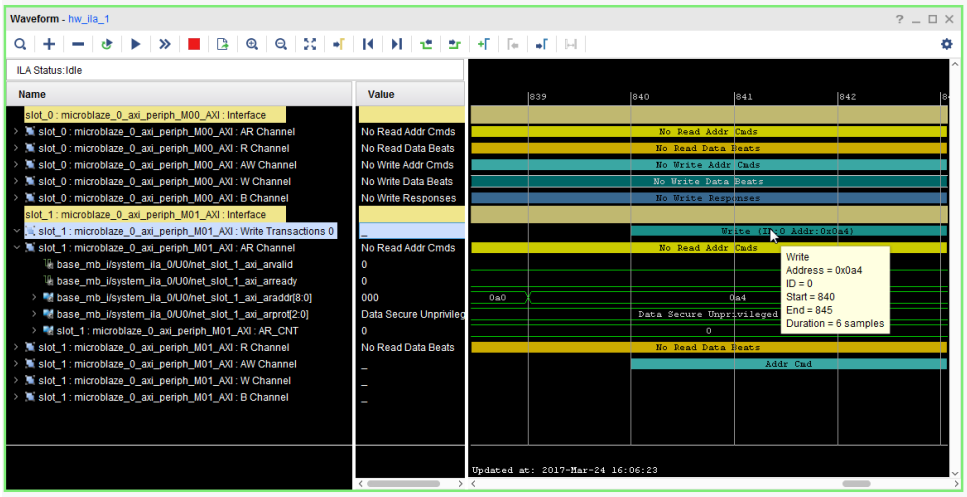
此波形可报告 AXI 接口相关传输事务、读取事件、写入事件、地址事件以及数据通道事件。

AXI 传输事务

AXI 传输事务用于报告 AXI 读取地址通道、AXI 读取数据通道、写入地址通道和写入数据通道的读写传输事务。

在“Waveform”（波形）窗口中将光标悬停于特定读取或写入传输事务上时，即可出现如下窗口，其中高亮显示与特定传输事务关联的“Address”（地址）、“ID”、“Start”（开始时间）、“End”（结束时间）和“Duration”（持续时间）：

图 143：**AXI 传输事务**



AXI 通道事件

“AXI Channel Events”（AXI 通道事件）组可用于报告 AXI 读取地址 (AR)、读取数据 (R)、写入地址 (AW)、写入数据 (W) 和写入响应 (B) 通道中的 AXI 事件。

读取地址 (AR) 通道事件

名称	描述
“No Read Addr Cmds”（无读取地址命令）	表示在“Read Address”（读取地址）通道上未发生任何地址命令事件。
“Addr Cmd”（地址命令）	表示读取传输事务存在有效的地址命令阶段。当 ARVALID = 1 时，会启动此事件。在以下情况下，会终止此事件：在相同时钟周期内 ARVALID = 1 且 ARREADY = 1。

读取地址通道信号组

此信号组由参与“Read Address Channel Event”（读取地址通道事件）的所有信号组成。这些信号如下所示：

- 信号线名称
  - ARVALID
  - ARREADY

- ARID
- ARADDR
- ARBURST
- ARLEN
- ARSIZE
- ARCACHE
- ARPROT
- ARLOCK
- ARQOS
- AR_CNT

读取数据通道事件

名称	描述
“No Read Data Beats”（无读取数据节拍）	表示在“Read Data”（读取数据）通道上未发生任何事件
“Data Beat”（数据节拍）	表示某一读取传输事务存在单个（非最后一个）数据节拍。在以下情况下会发生此事件：在当前时钟周期内 RAVLID = 1 且 RREADY = 1。
“Last Data”（最后数据）	表示某一读取传输事务的最后一个数据节拍。在以下情况下会发生此事件：在当前时钟周期内 RVALID = 1、RREADY = 1 且 RLAST = 1。

读取数据通道信号组

此信号组由参与“Read Data Channel Event”（读取数据通道事件）的所有信号组成。这些信号如下所示：

- 信号线名称
  - RVALID
  - RREADY
  - RLAST
  - RID
  - RDATA
  - RRESP
  - R_CNT

写入地址通道事件

名称	描述
“No Write Addr Cmds”（无写入地址命令）	表示在“Write Address”（写入地址）通道上未发生任何地址命令事件。

名称	描述
“Addr Cmd”（地址命令）	表示写入传输事务存在有效的地址命令阶段。当 AWVALID = 1 时，会启动此事件。在以下情况下，会终止此事件：在相同时钟周期内 AWVALID = 1 且 AWREADY = 1。

写入地址通道信号组

此信号组由参与“Write Address Channel Event”（写入地址通道事件）的所有信号组成。这些信号如下所示：

- 信号线名称
  - AWVALID
  - AWREADY
  - AWID
  - AWADDR
  - AWBURST
  - AWLEN
  - AWSIZE
  - AWCACHE
  - AWPROT
  - AWLOCK
  - AWQOS
  - AW_CNT

写入数据通道事件

名称	描述
“No Write Data Beats”（无写入数据节拍）	表示在“Write Data”（写入数据）通道上未发生任何事件。
“Data Beat”（数据节拍）	表示某一写入传输事务存在单个（非最后一个）数据节拍。在以下情况下会发生此事件：在当前时钟周期内 WVALID = 1 且 WREADY = 1，并且在上一个时钟周期内这 2 个信号中任一信号为“0”。
“Last Data”（最后数据）	表示某一写入传输事务的最后一个数据节拍。在以下情况下会发生此事件：在当前时钟周期内 WVALID = 1、WREADY = 1 且 WLAST = 1。

写入数据通道信号组

此信号组由参与“Write Data Channel Event”（写入数据通道事件）的所有信号组成。这些信号如下所示：

- 信号线名称
  - WVALID
  - WREADY
  - WLAST

- WDATA
- WSTRB

写入响应通道事件

名称	描述
“No Write Responses”（无写入响应）	表示在“Write Response”（写入响应）通道上未发生任何事件。
“Write Response”（写入响应）	表示写入传输事务的响应阶段。在以下情况下会发生此事件：在当前时钟周期内 BVALID = 1 且 BREADY = 1。

写入响应通道信号组

此信号组由参与“Write Response Channel Event”（写入响应通道事件）的所有信号组成。这些信号如下所示：

· 信号线名称
  - BVALID
  - BREADY
  - BID
  - BRESP
  - B_CNT

触发 AXI 地址命令和数据节拍

调试 AXI 接口通常涉及触发如下三种特定类型的 AXI 事件：地址命令结束、数据节拍结束以及写入响应。通常必须在不同接口通道上触发以上 1 项或多项事件。例如，要实现“读取地址命令结束或写入地址命令结束”的触发条件，需采用如下公式：

$$\text{Trigger Condition} = (((\text{ARVALID} == 1) \&\& (\text{ARREADY} == 1)) \,||\, ((\text{AWVALID} == 1) \&\& (\text{AWREADY} == 1)))$$

但这需要“乘积和 (SOP)”式布尔公式，而当所需 AXI 信号（例如 ARVALID 和 ARREADY）驻留在不同探针端口上时，则无法实现此类公式。为帮助完成此类触发，所需的 *VALID、*READY 和 *LAST 控制信号被串联在一起并连接到单一探针端口，如下表所示。

表 14：**AXI 通道控制信号探针**

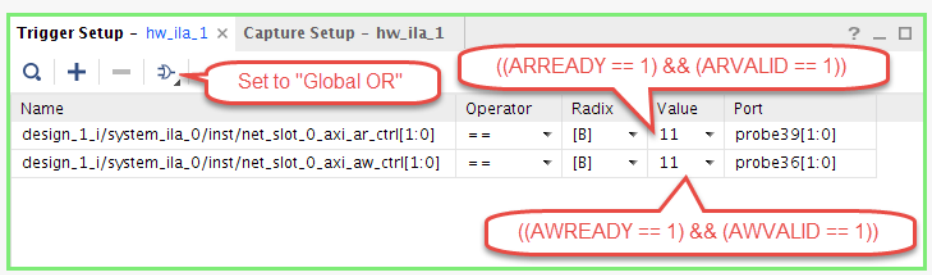
描述	探针名称	位 2	位 1	位 0
读取地址通道控制信号	*ar_ctrl[1:0]	不适用	ARREADY	ARVALID
读取数据通道控制信号	*r_ctrl[2:0]	RLAST	RREADY	RVALID
写入地址通道控制信号	*aw_ctrl[1:0]	不适用	AWREADY	AWVALID
写入数据通道控制信号	*w_ctrl[2:0]	WLAST	WREADY	WVALID
写入响应通道控制信号	*b_ctrl[1:0]	不适用	BREADY	BVALID

下表显示了如何使用单一 AXI 控制信号探针和 AXI 通道控制探针来实现实用的基本触发和捕获控制公式。下图显示了如何使用基本触发器设置 GUI 来实现“读取地址命令结束或写入地址命令结束”事件。

表 15：**触发和/或捕获控制事件**

AXI 事件	各 AXI 控制信号	组合的 AXI 通道控制探针
读取地址命令结束	((ARVALID == 1) && (ARREADY == 1))	(*ar_ctrl == 2'b11)
最后一个读取数据节拍结束	((RVALID == 1) && (RREADY == 1) && (RLAST == 1))	(*r_ctrl == 3'b111)
（非最后一个）读取数据节拍结束	((RVALID == 1) && (RREADY == 1) && (RLAST == 0))	(*r_ctrl == 3'b011)
写入地址命令结束	((AWVALID == 1) && (AWREADY == 1))	(*aw_ctrl == 2'b11)
写入读取数据节拍结束	((WVALID == 1) && (WREADY == 1) && (WLAST == 1))	(*w_ctrl == 3'b111)
（非最后一个）读取数据节拍结束	((WVALID == 1) && (WREADY == 1) && (WLAST == 0))	(*w_ctrl == 3'b011)

图 144：**“读取地址命令结束或写入地址命令结束”事件的基本触发器设置**



设置 VIO 核以执行测量

您添加到自己的设计中的 VIO 核会显示在“Hardware”（硬件）窗口中的目标器件下。如果未显示这些 VIO 核，请右键单击器件并选择“Refresh Hardware”（刷新硬件）。这样会重新扫描 FPGA 或自适应 SoC 并刷新“Hardware”（硬件）窗口。

注释：如果烧录和/或刷新 FPGA 或自适应 SoC 后仍未显示 VIO 核，请检查并确保已使用正确的 `.pdi` 文件完成器件烧录，并确认已实现的设计包含 VIO 核。此外，还请检查并确认有相应的 `.ltx` 探针文件（与 `.bit` 文件相匹配）与该器件关联。

单击 VIO 核（下图中名为 hw_vio_1 的核），以在“VIO Core Properties”（VIO 核属性）窗口中查看其属性。选中 VIO 核后，还应在“Debug Probes”（调试探针）窗口和 Vivado IDE 工作空间中对应的“VIO Dashboard”（VIO 仪表板）中看到对应于此 VIO 核的探针。

图 145：**“Hardware”窗口中的 VIO 核**

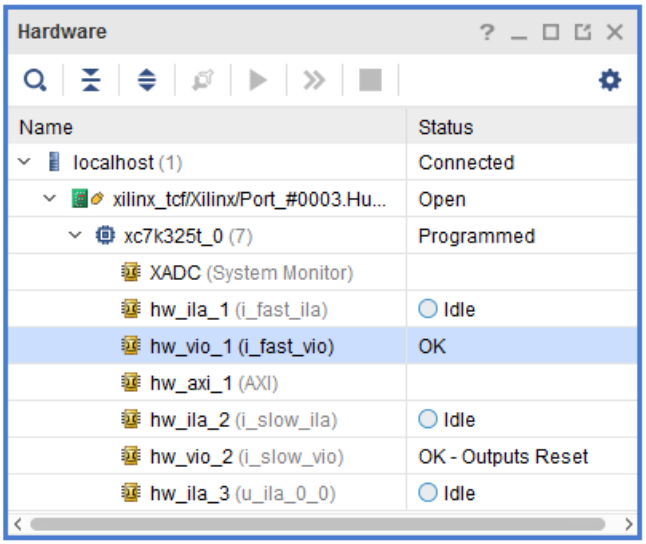
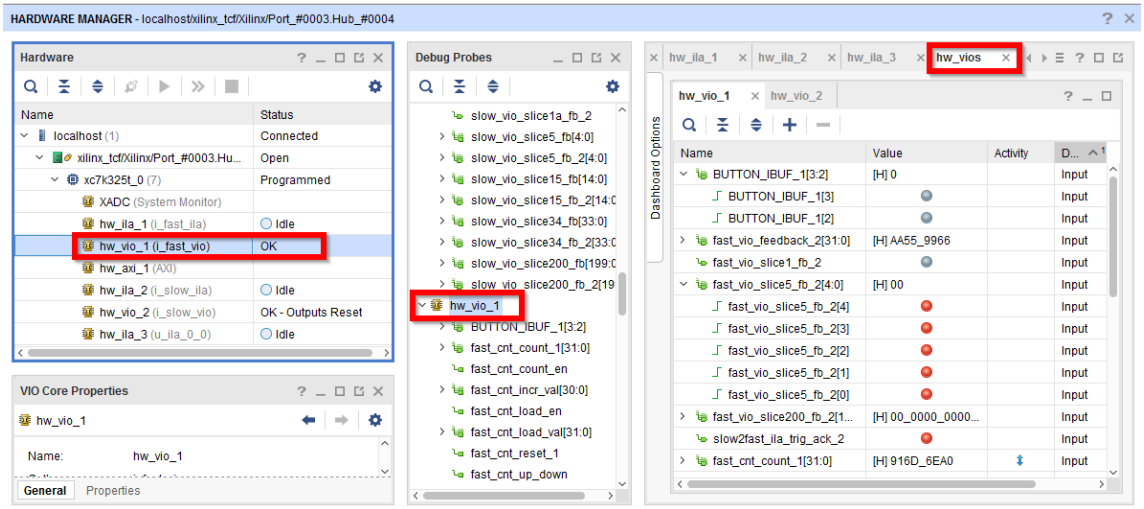


图 146：**各种视图中选中的 VIO 核**



VIO 核可能会与 Vivado IDE 不同步。请参阅“查看 VIO 核状态”，以获取有关如何解读 VIO 状态指示的更多信息。

VIO 核根据基于对象属性的置位/落实和刷新/获取模型来操作：

- 要读取 VIO 输入探针值，请首先以 VIO 核数值刷新 hw_vio 对象。获取对应 hw_probe 对象的属性值，以观察输入探针值。如需了解更多信息，请参阅 与 VIO 核输入探针进行交互。
- 要编写 VIO 输出探针值，请首先在 hw_probe 对象上获取期望的值作为属性。这些属性值将落实到硬件中的 VIO 核，以将这些值写入该核的输出探针端口。如需了解更多信息，请参阅 与 VIO 核输入探针进行交互。

相关信息

查看 VIO 核状态
与 VIO 核输入探针进行交互
与 VIO 核输出探针进行交互

查看 VIO 核状态

VIO 核可具有 0 个或更多个输入探针以及 0 个或更多个输出探针。

注释：VIO 核必须包含至少 1 个输入探针或输出探针。

“Hardware”（硬件）窗口中所示 VIO 核状态用于表示 VIO 核输出探针的当前状态。下表中描述了可能的状态值以及您需要采取的任何操作。

表 16：**VIO 核状态和所需的用户操作**

VIO 状态	描述	所需用户操作
OK – Outputs Reset	VIO 核输出与 Vivado IDE 同步，且输出处于初始状态或“reset”（复位）状态。	无
OK	VIO 核输出与 Vivado IDE 同步，但输出并未处于初始状态或“reset”状态。	无
Outputs out-of-sync	VIO 核输出与 Vivado IDE 不同步。	您必须选择以下 2 项用户操作之一： 在“Hardware”窗口中，右键单击 VIO 核并选中“Commit VIO Core Outputs”（落实 VIO 核输出）选项，以便将值从 Vivado IDE 写入 VIO 核。 在“Hardware”窗口中，右键单击 VIO 核并选中“Refresh Input and Output Values from VIO Core”（刷新来自 VIO 核的输入和输出值）选项，以便使用 VIO 核输出探针端口的当前值来更新 Vivado IDE。

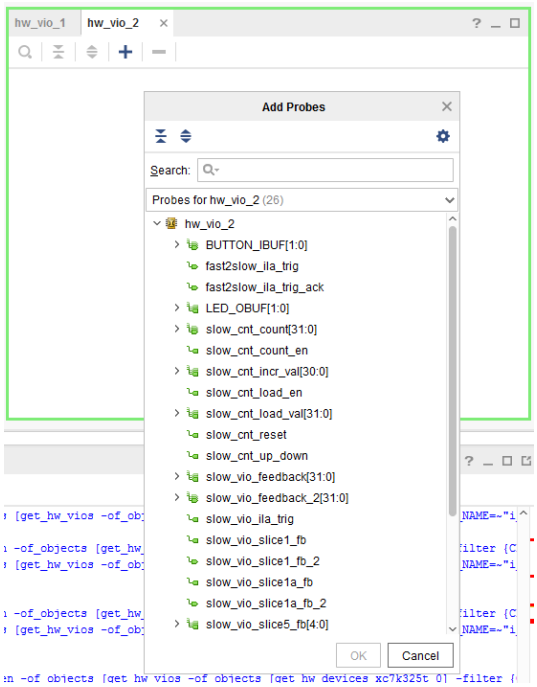
在“Debug Probes”窗口中查看 VIO 核

“VIO Dashboard”（VIO 仪表板）窗口中的“+”按钮用于查看、添加和删除属于 VIO 核的调试探针。

使用 VIO 仪表板

VIO 默认仪表板初始为空，您可向其中添加 VIO 探针，如下图显示。

图 147：**VIO 默认仪表板**

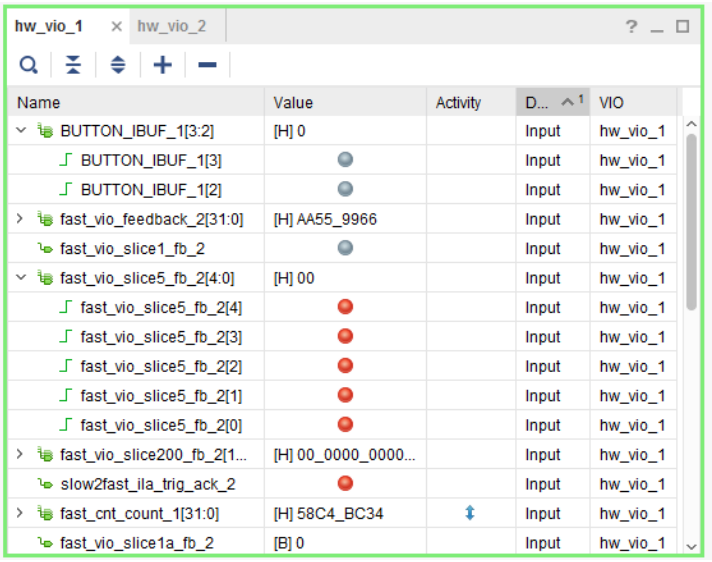


VIO 仪表板是给定 VIO 核相关的所有状态和控制信息的集中显示位置。刷新硬件器件并首次检测到 VIO 核时，将自动打开该核的 VIO 仪表板。如果需要手动打开或重新打开此仪表板，请在“Hardware”（硬件）窗口或“Debug Probes”（调试探针）窗口中右键单击此 VIO 核对象，然后选择“Open Dashboard”（打开仪表板）。

与 VIO 核输入探针进行交互

VIO 核输入探针用于从实际硬件的 FPGA 或自适应 SoC 内运行的设计中读取相应的值。VIO 输入探针通常用作为受测设计的状态指示器。您需要将 VIO 调试探针手动添加到 VIO 仪表板的“VIO Probes”（VIO 探针）窗口中。如需了解具体方法，请参阅以下章节：在“Debug Probes”窗口中查看 VIO 核。下图中显示了 VIO 仪表板的“VIO Probes”窗口中的 VIO 输入探针示例。

图 148：**核输入探针**



相关信息

在“Debug Probes”窗口中查看 VIO 核

使用 VIO 核视图读取 VIO 输入

VIO 输入探针可使用“VIO Dashboard”（VIO 仪表板）窗口的“VIO Probes”（VIO 探针）窗口来查看。每个输入探针均可作为表格中的单独一行来查看。VIO 输入探针的值显示在表中的“Value”（值）列中（请参阅“与 VIO 核输入探针进行交互”）。VIO 核输入值将根据 VIO 核的刷新率值定期更新。您可通过更改“VIO Properties”（VIO 属性）窗口中的“Refresh Rate (ms)”（刷新率 (ms)）或者通过运行以下 Tcl 命令来设置刷新率：

```
set_property CORE_REFRESH_RATE_MS 1000 [get_hw_vios hw_vio_1]
```

注释：将刷新率设为 0 会导致 VIO 核停止自动刷新。

注释：刷新率值过低可能导致 Vivado IDE 迟滞。

建议：AMD 建议将刷新率设为不低于 500 ms。

如果要手动读取 VIO 输入探针值，可使用 Tcl 命令。例如，如果要刷新并获取 VIO 核 hw_vio_1 的输入探针值（名为 BUTTON_IBUF），请运行以下 Tcl 命令：

```
refresh_hw_vio [get_hw_vios {hw_vio_1}] get_property INPUT_VALUE
[get_hw_probes BUTTON_IBUF]
```

相关信息

与 VIO 核输入探针进行交互

设置 VIO 输入显示类型和基数

VIO 输入探针的显示类型可通过如下方式进行设置：在“VIO Dashboard”（VIO 仪表板）窗口的“VIO Probes”（VIO 探针）窗口中，右键单击 VIO 输入探针，并选择：

- “Text”（文本），用于将输入显示为文本字段。这是对应 VIO 输入探针矢量（位宽大于 1）的唯一显示类型。
- “LED”，用于将输入显示为发光二极管 (LED) 的图形表示法。此显示类型仅适用于 VIO 输入探针标量以及 VIO 输入探针矢量的个别元素。您可将高低值设置为以下 4 种颜色中的任何颜色：
  - 灰（熄灭）
  - 红
  - 绿
  - 蓝

  当 VIO 输入探针的显示类型设置为“Text”时，您可通过如下方式更改基数：在“VIO Dashboard”窗口的“VIO Probes”窗口中，右键单击 VIO 输入探针，并选择：
- “Radix > Binary”（基数 > 二进制），用于将基数更改为二进制。
- “Radix > Octal”（基数 > 八进制），用于将基数更改为八进制。
- “Radix > Hex ”（基数 > 十六进制），用于将基数更改为十六进制。
- “Radix > Unsigned”（基数 > 无符号），用于将基数更改为无符号十进制。
- “Radix > Signed”（基数 > 有符号），用于将基数更改为有符号十进制。

您还可使用 Tcl 命令来设置 VIO 输入探针的基数。例如，要更改名为“BUTTON_IBUF”的 VIO 输入探针的基数，请运行以下 Tcl 命令：

```
set_property INPUT_VALUE_RADIX HEX [get_hw_probes BUTTON_IBUF]
```

观察和控制 VIO 输入活动

除了从 VIO 输入探针读取值外，您还可以监控 VIO 输入探针的活动。活动检测器可用于指示 Vivado IDE 定期更新期间 VIO 输入值何时发生改变。

VIO 输入探针活动值在“VIO Dashboard”（VIO 仪表板）窗口的“VIO Probes”（VIO 探针）窗口的活动列中显示为箭头：

- 向上箭头表示在活动持续期间，输入探针值已从 0 转变为 1。
- 向下箭头表示在活动持续期间，输入探针值已从 1 转变为 0。
- 双向箭头表示在活动持续期间，输入探针值已至少一次从 1 转变为 0 并从 0 转变为 1。

如需控制输入活动状态显示的持续时间，请在“VIO Dashboard”窗口的“VIO Probes”窗口中右键单击 VIO 输入探针并选择：

- “Activity Persistence > Infinite”（活动持续时间 > 无限），这样即可累积并保留活动值直至复位。
- “Activity Persistence > Long (80 samples)”（活动持续时间 > 长时间（80 个样本）），这样即可长时间累积并保留活动。
- “Activity Persistence > Short (8 samples)”（活动持续时间 > 短时间（8 个样本）），这样即可短时间累积并保留活动。

您还可使用 Tcl 命令来设置活动持续状态。例如，要将名为 BUTTON_IBUF 的 VIO 输入探针上的活动持续时间更改为较长的时间段，请运行以下 Tcl 命令：

```
set_property ACTIVITY_PERSISTENCE LONG [get_hw_probes BUTTON_IBUF]
```

要将任一给定核的所有输入探针的活动复位，请在“Hardware”窗口中右键单击 VIO 核，然后选择“Reset All Input Activity”（复位所有输入活动）。您也可以通过运行以下 Tcl 命令来执行此操作：

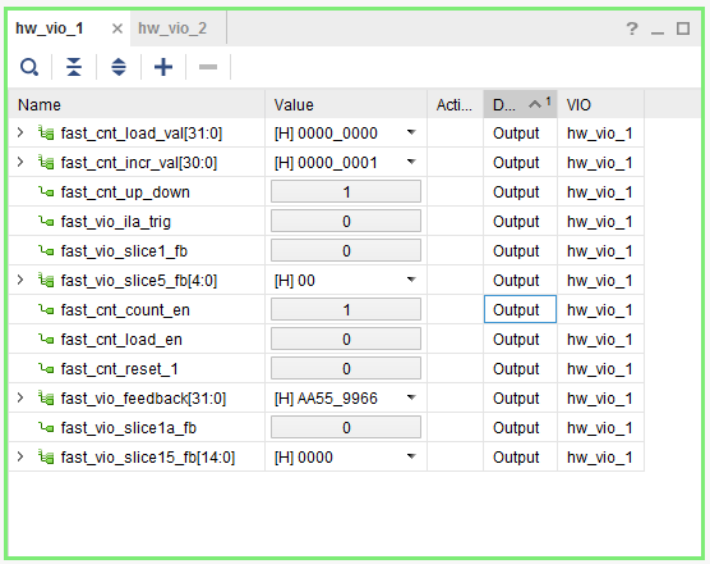
```
reset_hw_vio_activity [get_hw_vios {hw_vio_1}]
```

提示：要更改任一 VIO 输入探针矢量的多个标量成员的类型、基数和/或活动持续时间，请右键单击整个探针或者该探针的多个成员，然后从弹出菜单中进行选择。菜单选项适用于选中的所有探针标量。

与 VIO 核输出探针进行交互

VIO 核输出探针用于将值写入实际硬件的 FPGA 或自适应 SoC 内运行的设计中。VIO 输出探针通常用作为受测设计的低带宽控制信号。您需要将 VIO 调试探针手动添加到 VIO 仪表板的“VIO Probes”（VIO 探针）窗口中。如需了解具体方法，请参阅以下章节：在“Debug Probes”窗口中查看 VIO 核。下图中显示了 VIO 仪表板的“VIO Probes”窗口中的 VIO 输出探针示例。

图 149：**VIO 仪表板的“VIO Probes”窗口中 VIO 输出**



相关信息

在“Debug Probes”窗口中查看 VIO 核

使用 VIO 核视图来编写 VIO 输出

VIO 输出探针可使用“VIO Dashboard”（VIO 仪表板）窗口的“VIO Probes”（VIO 探针）窗口来进行设置。每个输出探针均可作为表格中的单独一行来查看。VIO 输出探针的值显示在表中的“Value”（值）列中（请参阅“与 VIO 核输出探针进行交互”）。只要在“Value”列中输入新的值，就会更新 VIO 核输出值。单击“Value”列即可显示下拉对话框。您可在“Value”文本字段中输入期望的值，然后单击“OK”。

也可以使用 Tcl 命令将新的值写出至 VIO 核。例如，如果要将二进制值“11111”写出至名为 vio_slice5_fb_2 且基数已设为 BINARY 的 VIO 输出探针，请运行以下 Tcl 命令：

```
set_property OUTPUT_VALUE 11111 [get_hw_probes vio_slice5_fb_2]
commit_hw_vio [get_hw_probes {vio_slice5_fb_2}]
```

相关信息

与 VIO 核输出探针进行交互

设置 VIO 输出显示类型和基数

VIO 输出探针的显示类型可通过如下方式进行设置：在“VIO Dashboard”（VIO 仪表板）窗口的“VIO Probes”（VIO 探针）窗口中，右键单击 VIO 输出探针，并选择以下任一项。

- Text（文本）：用于将输出显示为文本字段。这是对应 VIO 输入探针矢量（位宽大于 1）的唯一显示类型。
- Toggle Button（切换按钮）：用于将输出显示为切换按钮图形。此显示类型仅适用于 VIO 输出探针标量以及 VIO 输入探针矢量的个别元素。

当 VIO 输出探针的显示类型设置为“Text”时，您可通过如下方式更改基数：在“Debug Probes”（调试探针）窗口的“VIO Cores”（VIO 核）选项卡视图中，右键单击 VIO 输出探针并选择：

- “Radix” → “Binary”（基数 > 二进制），用于将基数更改为二进制。
- “Radix” → “Octal”（基数 > 八进制），用于将基数更改为八进制。
- “Radix” → “Hex”（基数 > 十六进制），用于将基数更改为十六进制。
- “Radix” → “Unsigned”（基数 > 无符号），用于将基数更改为无符号十进制。
- “Radix” → “Signed”（基数 > 有符号），用于将基数更改为有符号十进制。

您还可使用 Tcl 命令来设置 VIO 输出探针的基数。例如，要将名为“vio_slice5_fb_2”的 VIO 输出探针的基数更改为十六进制，请运行以下 Tcl 命令：

```
set_property OUTPUT_VALUE_RADIX HEX [get_hw_probes vio_slice5_fb_2]
```

复位 VIO 核输出值

VIO v2.0 核具有支持您为每个输出探针端口指定初始值的功能。您可在“Hardware”（硬件）窗口中右键单击 VIO 核并选择“Reset VIO Core Outputs”（VIO 核输出复位）选项来将 VIO 核输出探针端口复位为初始值。您还可使用 Tcl 命令来复位 VIO 核输出：

```
reset_hw_vio_outputs [get_hw_vios {hw_vio_1}]
```

注释：将 VIO 输出探针复位为其初始值会导致输出探针值与 Vivado IDE 出现不同步。请参阅“将 VIO 核输出值同步到 Vivado IDE”章节，以了解有关如何处理此类情况的信息。

相关信息

将 VIO 核输出值同步到 Vivado IDE

将 VIO 核输出值同步到 Vivado IDE

在复位 VIO 输出、对 FPGA 或自适应 SoC 进行重新烧录或者由其他 Vivado 工具实例设置输出值之后到启动当前实例之前，VIO 核的输出探针可能与 Vivado IDE 出现不同步。在上述任何情况下，只要 VIO 状态指示“Outputs out-of-sync”（输出不同步），就需要执行以下 2 项操作中的任一操作：

- 在“Hardware”窗口中，右键单击 VIO 核并选中“Commit VIO Core Outputs”（落实 VIO 核输出）选项，以便将值从 Vivado IDE 写入 VIO 核。您也可以运行以下 Tcl 命令来执行此操作：

```
commit_hw_vio [get_hw_vios {hw_vio_1}]
```

- 在“Hardware”窗口中，右键单击 VIO 核并选中“Refresh Input and Output Values from VIO Core”（刷新来自 VIO 核的输入和输出值）选项，以便使用 VIO 核输出探针端口的当前值来更新 Vivado IDE。您也可以运行以下 Tcl 命令来执行此操作：

```
refresh_hw_vio -update_output_values 1 [get_hw_vios {hw_vio_1}]
```

使用 JTAG-to-AXI Master 调试核进行硬件系统通信

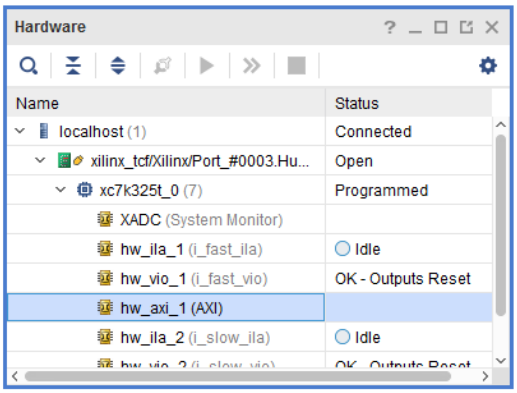
JTAG-to-AXI Master 调试核为可自定义核，可在运行时生成 AXI 传输事务并驱动 FPGA 内部的 AXI 信号。该核支持所有 AXI 和 AXI4-Lite 存储器映射接口，并且可支持位宽为 32 位或 64 位的数据接口。

您添加到设计中的 JTAG-to-AXI Master (JTAG-AXI) 核会显示在“Hardware”（硬件）窗口中的目标器件下。如果未显示这些 JTAG-AXI 核，请右键单击器件并选择“Refresh Hardware”（刷新硬件）。这样将重新扫描 FPGA 并刷新“Hardware”窗口。

注释：如果烧录和/或刷新 FPGA 器件后仍未显示 ILA 核，请检查并确保已使用正确的 `.bit` 文件完成器件烧录，并确认已实现的设计包含 ILA 核。

单击并选中此 JTAG-AXI 核（下图中名为 hw_axi_1 的核），以在“AXI Core Properties”（AXI 核属性）窗口中查看其属性。

图 150：**“Hardware”窗口中的 JTAG-to-AXI Master 核**



与硬件中的 JTAG-to-AXI Master 调试核进行交互

仅限 Tcl 命令可用于与 JTAG-to-AXI Master 调试核进行通信。您可使用 `create_hw_axi_txn` 命令和 `run_hw_axi` 命令来分别创建并运行 AXI 读取和写入传输事务。

复位 JTAG-to-AXI Master 调试核

创建并发出传输事务之前，重要的是使用以下 Tcl 命令来复位 JTAG-to-AXI Master 核：

```
reset_hw_axi [get_hw_axis hw_axi_1]
```

创建并运行读取传输事务

用于创建 AXI 传输事务的 Tcl 命令为 `create_hw_axi_txn`。如需了解有关如何使用此命令的更多信息，请在 Vivado IDE 的“Tcl console”（Tcl 控制台）中输入“`help create_hw_axi_txn`”。以下是有关如何从地址 0 创建 4 字 AXI 读取突发传输事务的示例：

```
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type READ -address
00000000 -len 4
```

其中：

- `read_txn` 是用户定义的传输事务名称
- `[get_hw_axis hw_axi_1]` 会返回 hw_axi_1 对象
- `-address 00000000` 是起始地址
- `-len 4` 会将 AXI 突发长度设置为 4 个字

下一步是运行先前使用 `run_hw_axi` 命令创建的传输事务。操作方式如下：

```
run_hw_axi [get_hw_axi_txns read_txn]
```

最后一步是获取通过运行该传输事务所读取的数据。您可使用 report_hw_axi_txn 或 `report_property` 命令在屏幕上打印此数据，或者也可以使用 `get_property` 命令来返回值，以供在别处使用。

```
report_hw_axi_txn [get_hw_axi_txns read_txn]
0 00000000 00000000
8 00000000 00000000
report_property [get_hw_axi_txns read_txn]
Property  Type  Read-only Visible Value
CLASS     string true     true    hw_axi_txn
CMD.ADDR  string false    true    00000000
CMD.BURST enum   false    true    INCR
CMD.CACHE int    false    true    3
CMD.ID    int    false    true    0
CMD.LEN   int    false    true    4
CMD.SIZE  enum   false    true    32
DATA      string false    true    00000000000000000000000000000000
HW_AXI    string true     true    hw_axi_1
NAME      string true     true    read_txn
TYPE      enum   false    true    READ
```

创建和运行写入传输事务

以下是有关如何从地址 0 创建 4 字 AXI 写入突发传输事务的示例：

```
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type WRITE -address
00000000 \        -len 4 -data {11111111_22222222_33333333_44444444}
```

其中：

- `write_txn` 是用户定义的传输事务名称
- `[get_hw_axis hw_axi_1]` 会返回 hw_axi_1 对象
- `-address 00000000` 是起始地址
- `-len 4` 会将 AXI 突发长度设置为 4 个字
- `-data {11111111_22222222_33333333_44444444}` - “-data”方向为左侧 LSB（即，地址 0）和右侧 MSB（即，地址 3）。

下一步是运行先前使用 `run_hw_axi` 命令创建的传输事务。操作方式如下：

```
run_hw_axi [get_hw_axi_txns write_txn]
```

重要提示！如果您对器件进行重新烧录，那么将删除所有现有 jtag_axi 传输事务。您可以重新创建这些传输事务。

提示：`run_hw_axi` Tcl 命令的可选实参 `-queue` 允许您以队列模式指定 hw_axi 传输事务。排队操作允许在 JTAG to AXI Master FIFO 中对最多 16 项读取和 16 项写入传输事务进行排队，并发出连续执行指令，以便降低传输事务之间的时延并提升性能。非排队传输事务则直接在提交时运行。

在实验室环境中使用 Vivado Logic Analyzer

Vivado Logic Analyzer 功能已集成到 Vivado IDE 和 Vivado Lab Edition 中。要使用 Vivado Logic Analyzer 功能对实验室环境内的目标开发板上运行的设计进行调试，需执行以下 3 项操作之一：

- 在实验室机器上安装并运行 Vivado Lab Edition。如需了解更多详情，请参阅本用户指南的 Vivado Lab Edition 部分。
- 在实验室机器上安装并运行完整版 Vivado IDE。
- 在远程实验室机器上安装最新版 Vivado Design Suite 或 Vivado Hardware Server (Standalone)，并在本地机器上使用 Vivado Logic Analyzer 功能连接至 Vivado Hardware Server (hw_server) 的远程实例。

相关信息

Vivado Lab Edition

连接到实验室机器上运行的远程 hw_server

如果您可通过网络连接至实验室机器，那么也可以通过连接至该远程实验室机器上运行的硬件服务器来连接到目标开发板。出于安全原因，在 Vivado IDE 中启动的 `hw_server` 仅侦听环回适配器（例如，`localhost` 或 `127.0.0.1`）。要执行远程访问，请手动启动 `hw_server`。

下列步骤解释了如何使用 Vivado Logic Analyzer 功能连接到实验室及其上运行的 Vivado 硬件服务器（Windows 平台上的 `hw_server.bat` 或 Linux 平台上的 `hw_server`）：

1. 在实验室机器上安装最新版本的 Vivado Design Suite 或 Vivado Hardware Server (Standalone) 单机版硬件服务器。

   重要提示！如果只需使用远程硬件服务器功能，那么无需在实验室机器上安装完整的 Vivado Design Suite 或 Vivado Lab Edition。但如果您想要在实验室机器上使用 Vivado 硬件管理器功能（例如，Vivado Logic Analyzer 或 Vivado Serial I/O Analyzer），则需在实验室机器上安装 Vivado Lab Edition。并且也无需任何软件许可证，即可运行硬件服务器、硬件管理器中的任意功能以及 Vivado Lab Edition。

2. 在远程实验室机器上启动 `hw_server` 应用。假定您已将 Vivado Hardware Server (Standalone) 安装至默认位置，并且您的实验室机器为 64 位 Windows 机器，请运行以下命令行：

   ```
   C:\Xilinx\VivadoHWSRV\vivado_release.version\bin\hw_server.bat
   ```

3. 在除实验室机器以外的其他机器上以 GUI 模式启动 Vivado IDE。
4. 遵循“连接至硬件目标并执行器件烧录”章节中的步骤，建立与已连接到实验室机器的目标开发板的连接。但请勿连接到 localhost 上运行的 Vivado CSE 服务器，而应改用实验室机器的主机名。
5. 遵循“设置 ILA 核以执行测量”章节及其后续章节中的步骤，对硬件内的设计进行调试。

相关信息

连接至硬件目标并执行器件烧录
设置 ILA 核以执行测量

硬件管理器 Tcl 对象和命令的描述

您可使用 Tcl 命令与所测试的硬件进行交互。硬件可组织为一组分层式第一类 Tcl 对象（请参阅下表）。

表 17：**硬件管理器 Tcl 对象**

Tcl 对象	描述
hw_server	指代硬件服务器的对象。每个 hw_server 都可包含 1 个或多个与之关联的 hw_target 对象。
hw_target	指代 JTAG 线缆或开发板的对象。每个 hw_target 都可包含 1 个或多个与之关联的 hw_device 对象。
hw_device	指代 JTAG 链中器件的对象，包括 AMD FPGA 或自适应 SoC。每个 hw_device 都可包含 1 个或多个与之关联的 hw_ila 对象。
hw_ila	指代 AMD FPGA 或自适应 SoC 中 ILA 核的对象。每个 hw_ila 对象都只能包含 1 个与之关联的 hw_ila_data 对象。每个 hw_ila 对象都可包含 1 个或多个与之关联的 hw_probe 对象。
hw_ila_data	指代从 ILA 调试核上传的数据的对象。
hw_probe	指代 ILA 调试核的探针输入的对象。
hw_vio	指代 AMD FPGA 或自适应 SoC 中 VIO 核的对象。

如需了解有关硬件管理器命令的更多信息，请在 Tcl 控制台中运行 `help -category hardware` Tcl 命令。

hw_server Tcl 命令的描述

下表包含用于与硬件服务器进行交互的所有 Tcl 命令的描述。

表 18：**hw_server Tcl 命令的描述**

Tcl 命令	描述
connect_hw_server	打开到硬件服务器的连接。
current_hw_server	获取或设置当前硬件服务器。
disconnect_hw_server	关闭到硬件服务器的连接。
get_hw_servers	获取硬件服务器名称列表。
refresh_hw_server	刷新到硬件服务器的连接。

hw_target Tcl 命令的描述

下表包含用于与硬件目标进行交互的所有 Tcl 命令的描述。

表 19：**hw_target Tcl 命令的描述**

Tcl 命令	描述
close_hw_target	关闭硬件目标。
current_hw_target	获取或设置当前硬件目标。
get_hw_targets	获取硬件服务器的硬件目标列表。
open_hw_target	打开到硬件服务器上的硬件目标的连接。
refresh_hw_target	刷新到硬件目标的连接。

hw_device Tcl 命令的描述

下表包含用于与硬件器件进行交互的 hw_device Tcl 命令的描述。

表 20：**hw_device Tcl 命令的描述**

Tcl 命令	描述
current_hw_device	获取或设置当前硬件器件。
get_hw_devices	获取目标的硬件器件列表。
program_hw_device	对 AMD FPGA 器件进行烧录。
refresh_hw_device	刷新硬件器件。

hw_ila Tcl 命令的描述

下表包含用于与 ILA 调试核进行交互的 hw_ila Tcl 命令的描述。

表 21：**hw_ila Tcl 命令的描述**

Tcl 命令	描述
current_hw_ila	获取或设置当前硬件 ILA。
get_hw_ilas	获取目标的硬件 ILA 列表。
reset_hw_ila	将 hw_ila 控制属性复位为默认值。
run_hw_ila	Arm® hw_ila 触发器。

表 21：**hw_ila Tcl 命令的描述** (续)

Tcl 命令	描述
wait_on_hw_ila	等待完成捕获所有数据。

hw_ila_data Tcl 命令的描述

下表包含用于与捕获的 ILA 数据进行交互的 hw_ila_data Tcl 命令的描述。

表 22：**hw_ila_data Tcl 命令的描述**

Tcl 命令	描述
current_hw_ila_data	获取或设置当前硬件 ILA 数据。
display_hw_ila_data	在波形查看器中显示 hw_ila_data。
get_hw_ila_data	获取 hw_ila_data 对象列表。
list_hw_samples	列出与各硬件探针关联的数据样本。
read_hw_ila_data	从文件读取 hw_ila_data。
upload_hw_ila_data	使 ILA 核停止捕获数据并上传所有已捕获的数据。
write_hw_ila_data	将 hw_ila_data 写入文件。

hw_probe Tcl 命令的描述

下表包含用于与捕获的 ILA 数据进行交互的所有 Tcl 命令的描述。

表 23：**hw_probe Tcl 命令的描述**

Tcl 命令	描述
create_hw_probe	基于物理 ILA 探针端口和/或常量值创建新的硬件探针。
delete_hw_probe	删除使用 `create_hw_probe` 命令创建的用户定义的硬件探针
get_hw_probes	获取硬件探针列表。

hw_vio Tcl 命令的描述

下表包含用于与 VIO 核进行交互的所有 Tcl 命令的描述。

表 24：**hw_vio Tcl 命令的描述**

Tcl 命令	描述
commit_hw_vio	将 hw_probe OUTPUT_VALUE 属性值写入 VIO 核。
get_hw_vios	获取 hw_vios 列表
refresh_hw_vio	以从 VIO 核读取的值更新 hw_probe INPUT_VALUE 和 ACTIVITY_VALUE 属性。
reset_hw_vio_activity	为与指定 hw_vio 对象关联的 hw_probes 复位 VIO ACTIVITY_VALUE 属性。
reset_hw_vio_outputs	将 VIO 核输出复位到初始值。

hw_axi 和 hw_axi_txn Tcl 命令的描述

下表包含用于与 JTAG-to-AXI Master 核进行交互的所有 Tcl 命令的描述。

表 25：**hw_axi 和 hw_axi_txn Tcl 命令的描述**

Tcl 命令	描述
create_hw_axi_txn	创建硬件 AXI 传输事务对象。
delete_hw_axi_txn	删除硬件 AXI 传输事务对象。
get_hw_axi_txns	获取硬件 AXI 传输事务对象列表。
get_hw_axis	获取硬件 AXI 对象列表。
refresh_hw_axi	刷新硬件 AXI 对象状态。
report_hw_axi_txn	报告已格式化的硬件 AXI 传输事务数据。
reset_hw_axi	复位硬件 AXI 核状态。
run_hw_axi	在对应 hw_axi 对象中运行硬件 AXI 读写传输事务并更新传输事务状态。

hw_sysmon Tcl 命令描述

下表包含与“System Monitor”（系统监控器）核进行交互的所有 Tcl 命令的描述。

表 26：**hw_sysmon Tcl 命令描述**

Tcl 命令	描述
commit_hw_sysmon	将 hw_sysmon 对象上定义的当前属性值落实到硬件器件上的系统监控器寄存器。
get_hw_sysmon_reg	返回指定的 hw_sysmon 对象的指定地址处定义的系统监控器寄存器的十六进制值。
get_hw_sysmons	返回当前硬件器件上定义的 Sysmon 调试核对象的列表。
refresh_hw_sysmon	使用来自当前 hw_device 的系统监控器上的值刷新 hw_sysmon 对象的属性。
set_hw_sysmon_reg	将位于指定地址的系统监控器寄存器设置为指定的十六进制值。

注释：在 Vivado Tcl 控制台上输入 <命令名> `-help` 即可获取有关上述每条命令的详细帮助信息。

使用 Tcl 命令来与 JTAG-to-AXI Master 核进行交互

以下示例提供了与下列系统示例进行交互的 Tcl 命令脚本：

- 1 条 KC705 评估板的 Digilent JTAG-SMT1 线（序列号 12345），可通过 localhost:3121 上运行的 Vivado hw_server 来访问。
- 在 KC705 评估板上的 XC7K325T 器件中运行的设计内包含单个 JTAG-to-AXI Master 核。
- JTAG-to-AXI Master 核位于基于 AXI 的系统中，此系统包含 1 个 AXI BRAM Controller Slave 核。

Tcl 命令脚本示例

```
# Connect to the Digilent Cable on localhost:3121 connect_hw_server -url
localhost:3121 current_hw_target [get_hw_targets */xilinx_tcf/Digilent/
12345] open_hw_target # Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0] refresh_hw_device -
update_hw_probes false [lindex [get_hw_devices] 0] set_property
PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0] set_property
PROBES.FILE {C:/design.ltx} [lindex [get_hw_devices] 0] program_hw_devices
[lindex [get_hw_devices] 0] refresh_hw_device [lindex [get_hw_devices] 0] #
Reset the JTAG-to-AXI Master core reset_hw_axi [get_hw_axis hw_axi_1] #
Create a read transaction bursts 128 words starting from address 0
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type read \ -address
00000000 -len 128 # Create a write transaction bursts 128 words starting at
address 0 # using a repeating fill value of
11111111_22222222_33333333_44444444 # (where LSB is to the left)
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type write \ -address
00000000 -len 128 -data {11111111_22222222_33333333_44444444} # Run the
write transaction run_hw_axi [get_hw_axi_txns write_txn] # Run the read
transaction run_hw_axi [get_hw_axi_txns read_txn]
```

使用 Tcl 命令来执行 ILA 测量

以下示例提供了与下列系统示例进行交互的 Tcl 命令脚本：

- 1 条 KC705 评估板的 Digilent JTAG-SMT1 线（序列号 12345），可通过 localhost:3121 上运行的 Vivado CSE 服务器来访问。
- 在 KC705 评估板上的 XC7K325T 器件中运行的设计内包含单个 ILA 核。
- ILA 核包含名为 counter[3:0] 的探针。

Tcl 命令脚本示例

```
# Connect to the Digilent Cable on localhost:3121 connect_hw_server -url
localhost:3121 current_hw_target [get_hw_targets */xilinx_tcf/Digilent/
12345] open_hw_target # Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0] refresh_hw_device -
update_hw_probes false [lindex [get_hw_devices] 0] set_property
PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0] set_property
PROBES.FILE {C:/design.ltx} [lindex [get_hw_devices] 0] program_hw_devices
[lindex [get_hw_devices] 0] refresh_hw_device [lindex [get_hw_devices] 0] #
Set Up ILA Core Trigger Position and Probe Compare Values set_property
CONTROL.TRIGGER_POSITION 512 [get_hw_ilas hw_ila_1] set_property
COMPARE_VALUE.0 eq4'b0000 [get_hw_probes counter] # Arm the ILA trigger and
wait for it to finish capturing data run_hw_ila hw_ila_1 wait_on_hw_ila
hw_ila_1 # Upload the captured ILA data, display it, and write it to a file
current_hw_ila_data [upload_hw_ila_data hw_ila_1] display_hw_ila_data
[current_hw_ila_data] write_hw_ila_data my_hw_ila_data [current_hw_ila_data]
```

Trigger At Startup

“Trigger at Start up”（启动时触发）功能用于在设计烧录文件（.bit 或 .pdi）中配置 ILA 核的触发器设置，完成 ILA 核的预装备，以便在器件启动后立即触发。只需将一般情况下应用于硬件设计中运行的 ILA 核的各项触发器设置都应用于已实现的设计中的 ILA 核即可完成此设置。

重要提示！以下使用“Trigger at Start up”的进程假定您的硬件中的 ILA 设计有效且正常运行，并且综合流程期间 ILA 核尚未平铺。

提示：以下示例参考了 FPGA 产品的比特流，但可将相同的步骤应用于针对 Versal 架构的 PDI。

要使用“Trigger at Start up”功能，请执行以下步骤：

1. 照常运行 ILA 流程直至成功完成首次直通，以设置触发器条件。
   a. 打开目标、配置器件并启动“ILA Dashboard”（ILA 仪表板）。
   b. 在 ILA Dashboard 中输入 ILA 核的触发器公式。
2. 在 Vivado Tcl 命令行中，导出 ILA 核的触发器寄存器映射文件。此文件包含所有寄存器设置，用于发回到已实现的网表上并“盖戳”。其输出是单一文件。

```
% run_hw_ila -file ila_trig.tas [get_hw_ilas hw_ila_1]
```

3. 返回并打开 Vivado IDE 中先前已实现并已布线的设计。根据工程流程，可通过 2 种方法来执行此操作。
   a. 工程模式：使用 Flow Navigator 打开已实现的设计。
   b. 非工程模式：打开已布线的检查点：%open_checkpoint <file>.dcp
4. 在“Implemented Design”（已实现的设计）Tcl 控制台上，将触发器设置应用于存储器中的当前设计，即您的已布线的网表。

```
%apply_hw_ila_trigger ila_trig.tas
```

   注释：如果出现 ERROR 指示 ILA 核在综合期间已平铺，那么您需要重新生成设计并强制综合，以便为 ILA 核保留层级。请确保您的硬件中的 ILA 设计有效且正常运行，并且综合流程期间 ILA 核尚未平铺。
5. 在实现的设计的 Tcl 控制台中，使用 write_bitstream 命令（针对 7 系列、UltraScale 和 UltraScale+ FPGA 或 SoC）或 write_device_image 命令（针对 Versal 器件）搭配“Trigger at Start up”（启动时触发）设置来编写新器件镜像。

   重要提示！要选择已布线的设计更改，必须在 Tcl 命令控制台中执行以下操作：write_bitstream trig_at_startup.bit
6. 返回“Hardware Manager”（硬件管理器），并使用上一步中生成的新 .bit 文件进行重新配置。您必须通过 GUI 或 Tcl 命令来为已更新的 .bit 文件设置属性。请确保将新的 .bit 文件设置为在硬件工具中执行配置时所用的文件。
   a. 在硬件树中选择器件。
   b. 指定步骤 5 中生成的 .bit 文件。
7. 使用新的 .bit 文件执行器件烧录。

   烧录完成后，应在启动时立即装备新的 ILA 核。您应可在 ILA 核的“Trigger Capture Status”（触发捕获状态）中看到相关指示信息。如果发生触发或捕获事件，则表示现在 ILA 核中已填入捕获的数据样本。

存储器校准调试

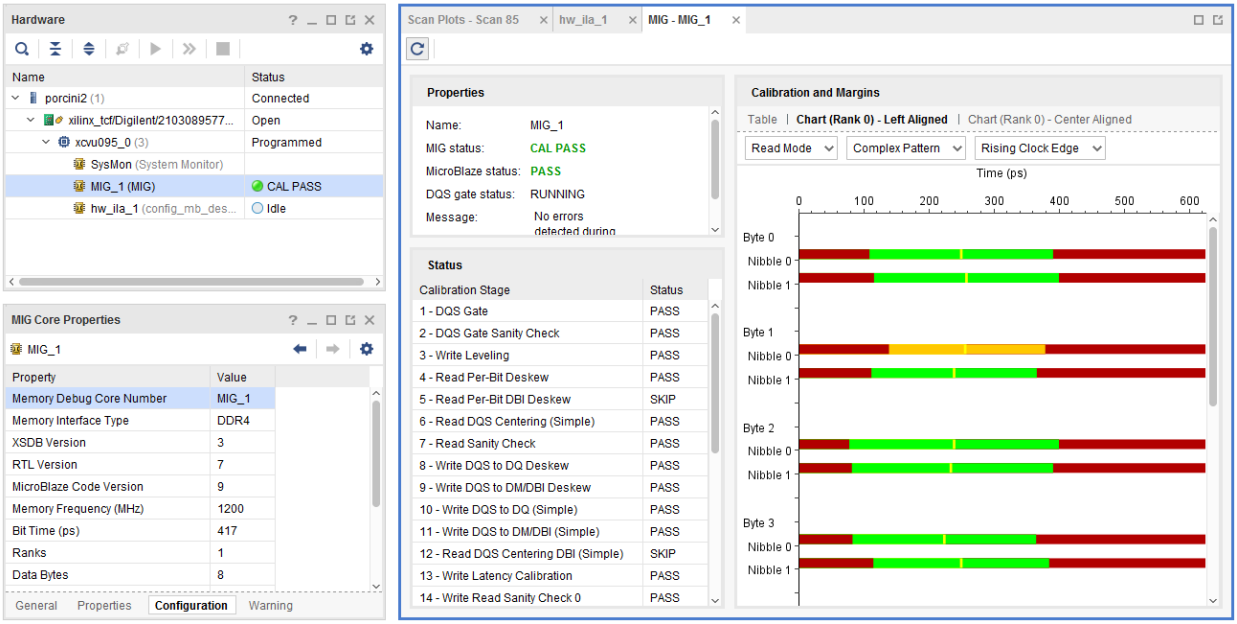
Vivado 中的存储器接口 IP 支持校准调试。其中存储了实用的核配置、校准和数据窗口信息，可在 Vivado 硬件管理器中访问这些信息。“Memory Calibration Debug”（存储器校准调试）可随时用于读取此信息，并从存储器接口 IP 中获取宝贵的统计数据和反馈。通过 Vivado 硬件管理器中的“Memory Calibration Debug”（存储器校准调试）GUI 或者通过可用的存储器校准调试 Tcl 命令即可查看这些信息。

存储器校准调试 GUI 使用方法

配置器件时，存储器接口会显示在 Vivado 硬件管理器中。

存储器校准内容显示在调试接口中，可用于快速识别校准状态和读写窗口裕度。此调试接口始终包含在生成的存储器接口（AMD UltraScale™ 和 AMD UltraScale+™）设计中。

图 151：**存储器校准调试接口**



存储器校准调试中的 Tcl 使用方法

连接到 Vivado 硬件管理器中的硬件时，在 Vivado Tcl 控制台中使用以下 Tcl 命令即可输出 Vivado IDE 中显示的所有存储器校准调试内容。

- `get_hw_migs`
  - 显示设计中存在的存储器接口。
- `refresh_hw_mig [lindex [get_hw_migs] 0]`
  - 仅刷新索引标明的存储器接口（索引从 0 开始）。
- `report_propery[lindex [get_hw_migs] 0]`

- 报告可供存储器接口使用的所有参数。
- 其中，0 是要报告的存储器接口的索引（索引从 0 开始）。

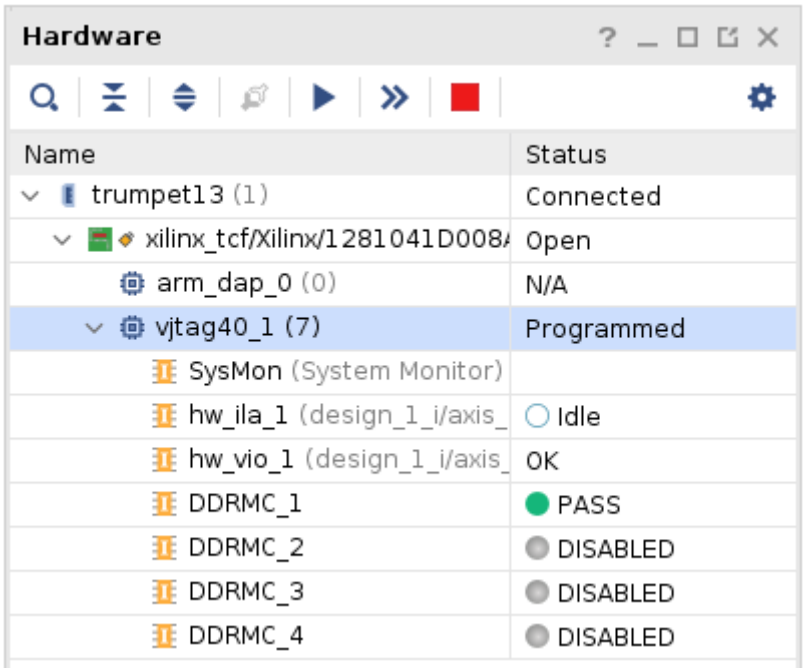
如需了解更多具体细节，请参阅以下文档中的 UltraScale 或 7 系列存储器校准调试命令：

- 答复记录 43879：MIG 7 系列 DDR3/DDR2 - 硬件调试指南。
- 《UltraScale 架构 FPGA 存储器 IP LogiCORE IP 产品指南》(PG150)。

DDRMC 校准调试中的 GUI 使用方法

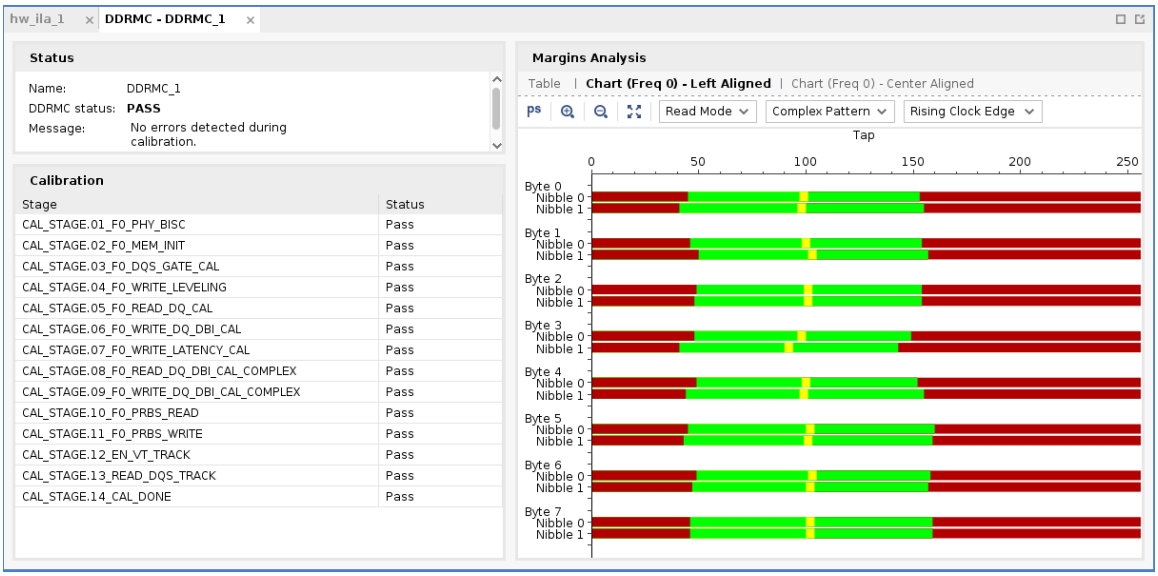
DDRMC 是 Versal 架构中包含的诸多集成块之一。配置 Versal 器件时，在 Vivado 硬件管理器中会同时显示已启用和已禁用的存储器接口。

图 152：**含 DDRMC 校准状态的硬件窗口**



存储器校准内容显示在调试接口中，可用于快速识别校准状态和读写窗口裕度。在 DDRMC 集成块中始终启用此调试接口。

图 153：**DDRMC 校准调试接口**



DDRMC 校准调试中的 Tcl 使用方法

连接到 Vivado 硬件管理器中的硬件时，在 Vivado Tcl 控制台中使用以下 Tcl 命令即可输出 Vivado IDE 中显示的 DDRMC 校准调试内容。

- `get_hw_ddrmcs`
  - 获取 Versal 自适应 SoC 集成的核以及 DDRMC 软核的列表。
- `refresh_hw_ddrmc [lindex [get_hw_ddrmcs] 0]`
  - 仅刷新索引标明的 DDRMC（索引从 0 开始）。
- `report_property [lindex [get_hw_ddrmcs] 0]`
  - 报告 DDRMC 可用的所有参数，其中 0 是要报告的 DDRMC 的索引（索引从 0 开始）。

如需了解有关 DDRMC 的更多信息，请参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

在 Vivado 硬件管理器中调试 Dynamic Function eXchange (DFX) 设计

Vivado 硬件管理器支持对 DFX 设计进行调试。为了成功对此类设计完成调试，需要先完成完整设计比特流烧录，然后才能对部分比特流进行烧录以替换特定可重配置模块。

要获取在 DFX 设计中例化调试核的示例和 Vivado 硬件管理器中该功能的相关信息，请参阅《Vivado Design Suite 教程：Dynamic Function eXchange》(UG947) 中的“Vivado 调试和 DFX 工程流程”章节。

高带宽存储器 (HBM) 监控器

某些 AMD Virtex™ UltraScale+™ FPGA 包含集成高带宽存储器 (HBM) 控制器和存储器栈。集成的 HBM 控制器和存储器栈包含性能计数器和温度传感器。HBM 监控器可随时用于实时访问、捕获和导出 HBM 裸片上的性能监控和温度传感器数据。

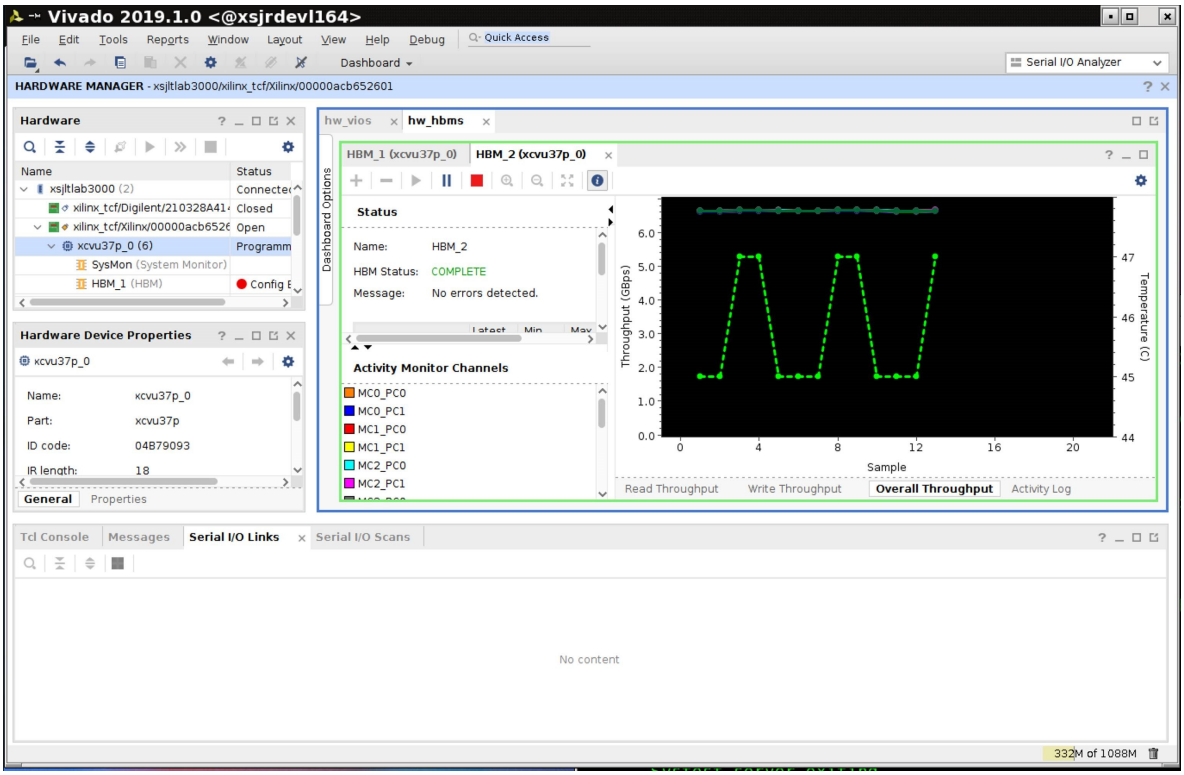
HBM 监控器中的 GUI 使用方法

如要配置启用 HBM 的器件并且其中设计包含 AXI High Bandwidth Memory Controller 实例，那么在 Vivado 硬件管理器中会显示 HBM 接口。

在生成的 High Bandwidth Memory Controller 中，始终包含针对 HBM 监控器的支持。HBM 监控器可显示栈温度、读取、写入和总体吞吐量。

您可将捕获的数据导出至逗号分隔值 (CSV) 格式的文本文件，以供后续进一步处理或分析。

图 154：**查看实时性能**



HBM 监控器中的 Tcl 使用方法

在 Vivado Tcl 控制台内，连接到 Vivado 硬件管理器中的硬件时，可使用以下 Tcl 命令与 HBM 监控器进行交互。

- `get_hw_hbms` - 显示设计中存在的 HBM 接口列表。
- `refresh_hw_hbm [lindex [get_hw_hbms] 0]` - 刷新一个或多个指定硬件 HBM 的状态，在此例中，即以索引 0 表示的 HBM。

- `report_property [lindex [get_hw_hbms] 0]` - 报告指定 HBM 接口可用的所有参数，在此例中，即以索引 0 表示的 HBM 接口。
- `run_hw_hbm_amon [lindex [get_hw_hbms] 0]` - 针对一个或多个指定硬件 HBM 启用活动监控器的运行。
- `stop_hw_hbm_amon [lindex [get_hw_hbms] 0]` - 针对一个或多个指定硬件 HBM 禁用活动监控器的运行。

如需了解具体详情并获取示例，请参阅《AXI High Bandwidth Controller LogiCORE IP 产品指南》(PG276) 附录 D。

PCI Express 链路调试

Vivado 中的 Versal PCI Express® 集成块支持链路调试。如果启用此选项，该核会存储 Vivado 硬件管理器内可访问的链路训练状态机 (LTSSM) 状态转换。

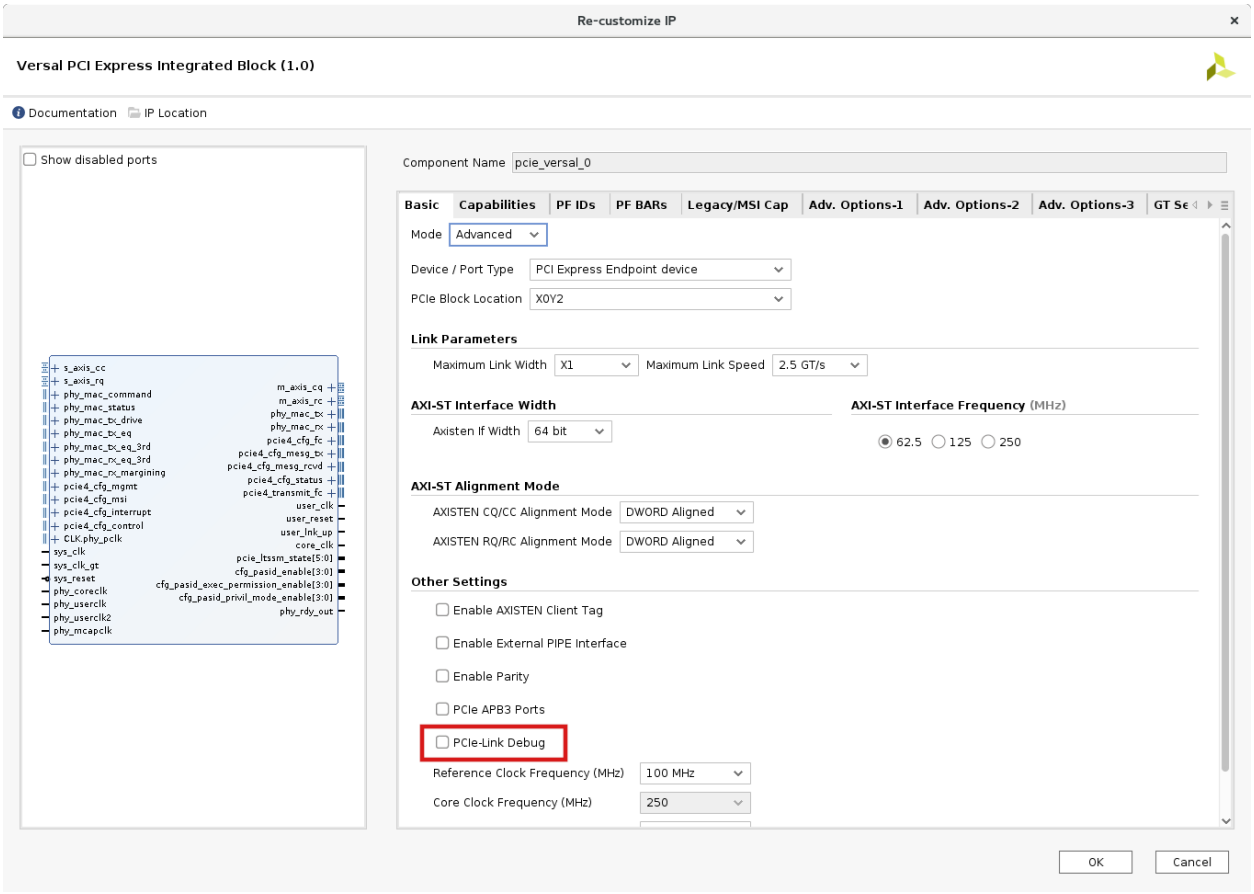
启用 PCI Express 链路调试

要使用 PCI Express 链路调试，必须在 Versal PCI Express Integrated Block IP 中将其启用。

要启用 PCI Express 链路调试功能，请执行以下操作：

1. 调用 Versal PCI Express Integrated Block IP 配置 GUI。
2. 在“Basic”（基础）选项卡下，将“Mode”（模式）更改为“Advance”（高级）。
3. 在“Other Settings”（其他设置）下，勾选“PCIe-Link Debug”（PCIe 链路调试）。

图 155：**在 Versal PCI Express Integrated Block IP 配置 GUI 中启用 PCI Express 链路调试**

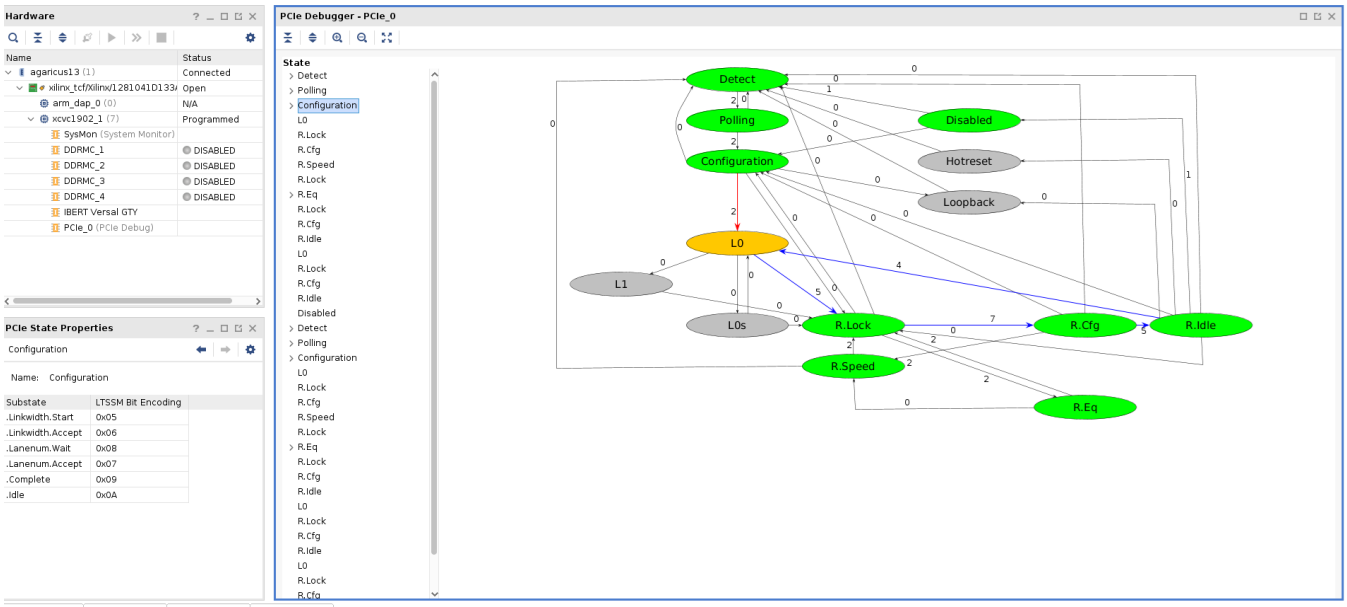


PCI Express 链路调试 GUI 使用方法

配置器件时如果启用了 PCI Express 核，那么这些核会显示在 Vivado 硬件管理器内。

PCI Express LTSSM 调试内容将显示在 LTSSM 状态转换图中。此界面可显示 LTSSM 状态转换的排序列表（其中显示已访问哪些状态），并可显示一份图表，以显示 LTSSM 中已访问的状态和当前占用的状态。

图 156：**PCI Express 链路调试界面**



ChipScoPy API

ChipScoPy 是来自 AMD 的开源工程，支持对硬件内运行的 AMD Versal™ 调试 IP 进行高级别控制，而无需使用 AMD Vivado™ 硬件管理器。开发者通过使用简单的 Python API 即可开发应用，用于与 ChipScope™ 调试 IP（例如，Integrated Logic Analyzer (ILA)、Virtual IO (VIO)、器件存储区访问等）进行通信并对其加以控制。

如需了解有关 ChipScoPy Python API 的更多信息，请参阅 https://github.com/Xilinx/chipscopy。

第 14 章

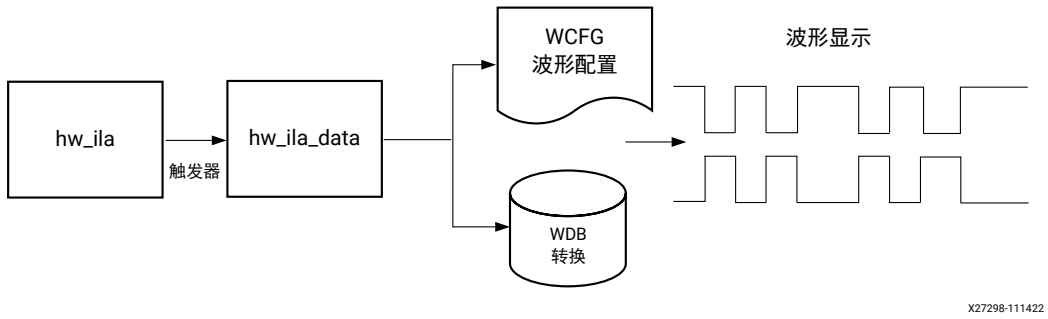
在波形查看器中查看 ILA 探针数据

AMD Vivado™ 集成设计环境 (IDE) 中的 ILA 波形查看器提供了一种强大的方法，用于分析从 ILA 调试核捕获的数据。成功触发 ILA 核并捕获数据后，Vivado 会使用从 ILA 核收集的数据自动填充对应的波形查看器。使用 Vivado 工程模式时，在不同 Vivado 会话之间，可配置波形设置（例如，颜色、基数选择和信号排序）将保持不变且方便记忆。

ILA 数据与波形关系

了解 hw_ila_data 捕获的 ILA 数据对象与波形之间的关系是很有用的，如下图所示。

图 157：**ILA 数据与波形关系**



`hw_ila` Tcl 对象表示硬件中的 ILA 核。每次 ILA 核上传捕获的数据后，这些数据都会存储在对应 Tcl `hw_ila_data` 对象的存储器中。这些对象按可预测的方式来命名，即硬件“`hw_ila_1`”中的首个 ILA 核在触发并上传后，会在名为“`hw_ila_data_1`”的对应 Tcl 数据对象中生成数据。联机处理硬件后，每个波形均由存储器内的 `hw_ila_data` 对象予以支持，并与此对象保持 1:1 对应关系，如上图中的图例所示。对于每个 Tcl `hw_ila_data` 对象，将在 Vivado 工程目录中创建并自动跟踪波形数据库 (WDB) 文件和波形配置 (WCFG) 文件。上图显示的数据流向为从左侧硬件 hw_ila 流至右侧显示的波形。

波形配置 WCFG 文件和波形转换数据库 WDB 文件的组合包含 Vivado 波形用户界面中显示的波形数据库和自定义设置。这些波形文件在 Vivado ILA 流程中自动进行管理，用户不应直接修改 WDB 或 WCFG 文件。波形配置可通过更改波形查看器中的对象（例如，信号颜色、总线基数、信号顺序、标记等）来进行修改。这样可将波形配置更改自动保存至 Vivado 工程中相应的 WCFG 文件。

Tcl 命令 `write_hw_ila_data` 可用于将波形配置和数据存档，以供后续查看。这样即可将 hw_ila_data、波形数据库和波形配置保存在存档中以便后续脱机查看。请参阅“保存和复原从 ILA 核捕获的数据”以获取有关如何使用 `read_hw_ila_data` 和 `write_hw_ila_data` 来脱机存储和检索波形的详细信息。

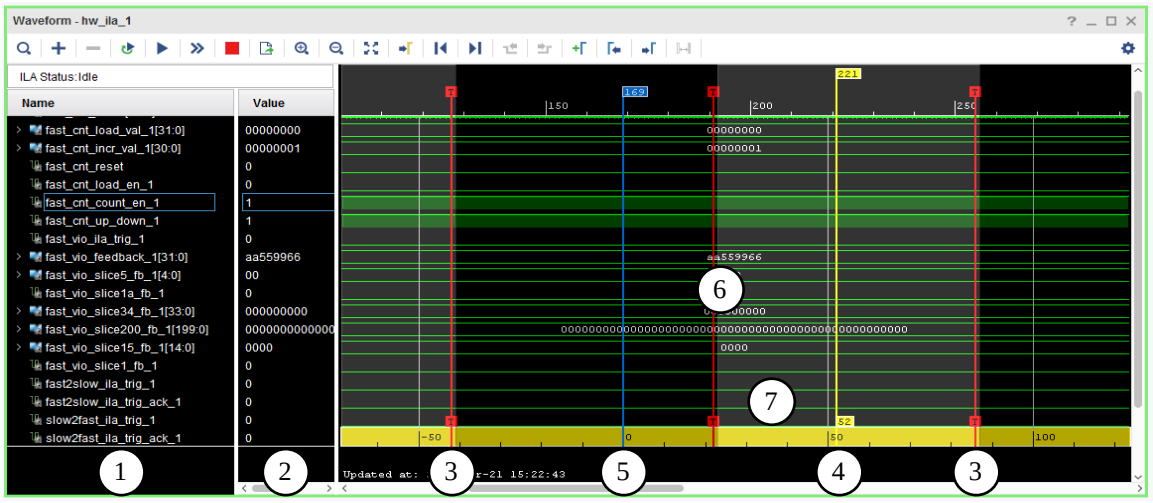
相关信息

保存和复原从 ILA 核捕获的数据

波形查看器布局

ILA 波形查看器（有时被称为波形配置）由多个动态对象组成，这些动态对象通过协同工作来为捕获的 ILA 数据提供完整的可视化工具，如下图所示。

图 158：**显示捕获的 ILA 数据的波形查看器**



X18959-032717

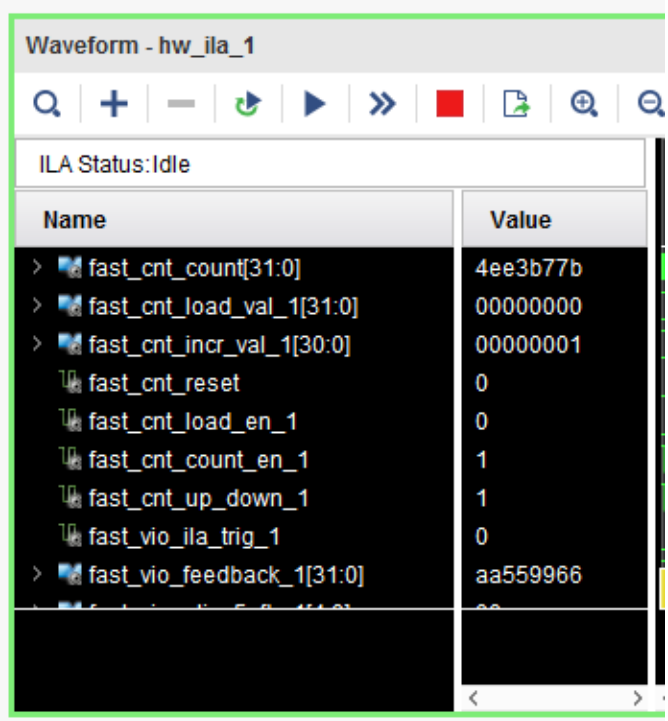
前图中带标签的对象描述如下：

1. 来自 ILA 探针文件 (`.ltx`) 的信号线或总线名称
2. 光标处的信号线或总线值
3. 触发器标记（红线）
4. 光标（黄线）
5. 标记（蓝线）
6. ILA 捕获窗口转换（交替清空/灰色区域）
7. 浮动测量标尺（黄色条形区域）

波形查看器操作

波形查看器的“Name”（名称）列中所示标量和总线表示波形中探针设计对象的名称（请参阅下图）。这些对象对应于 ILA 核的硬件探针（请参阅 `get_hw_probes` Tcl 命令）。

图 159：**波形查看器中所示的 ILA 探针名称和值**



首次触发并上传 ILA 数据后，就会立即在波形查看器中填充连接到该 ILA 核的所有探针。除了在查看器中移除现有探针或添加新探针外，还可以在其中对探针进行自定义。本章涵盖了波形查看器的基本操作。

从波形中移除探针

默认情况下，在首次执行触发和上传操作期间会将所有探针添加到波形中。如果您不希望波形包含所有探针，那么只需从查看器移除探针即可。

要从波形查看器移除探针，请在“Name”（名称）列中右键单击要删除的标量或总线，然后从弹出菜单中单击“Delete”（删除）。或者，也可以选中要删除的信号或总线，然后按 Delete 键。移除探针并非真的从存储器中删除探针转换数据，而是在视图中隐藏这些数据而已。

向波形中添加探针

要向波形中添加探针，请在“Debug Probes”（调试探针）窗口中选择要为相关 ILA 核添加的探针，右键单击，然后从弹出菜单中单击“Add Probes to Waveform”（向波形中添加探针）。

要向“Waveform”（波形）窗口添加另一个信号或总线副本，请在“Waveform”窗口中选中该信号或总线。依次单击“Edit”→“Copy”（编辑 > 复制）或者按 Ctrl+C。这样即可将所选对象复制到剪贴板中。依次单击“Edit”→“Paste”（编辑 > 粘贴）或者按 Ctrl+V 以在波形中粘贴此对象的副本。

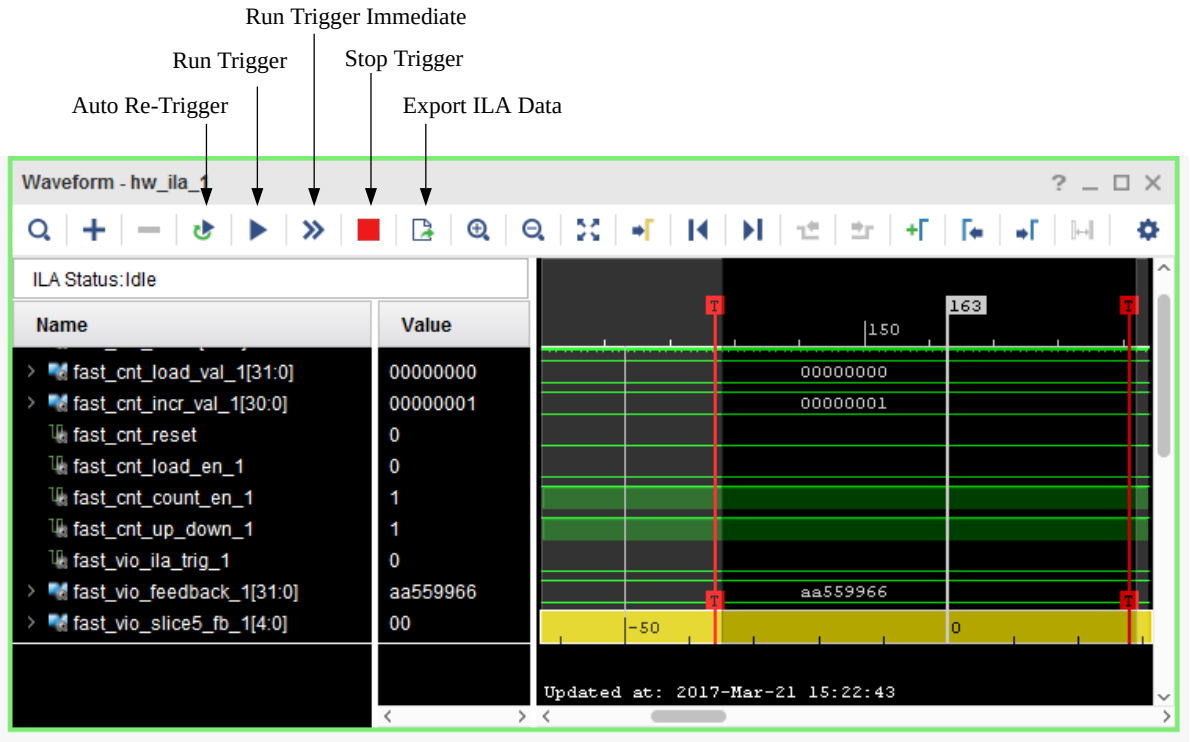
您也可以使用 Tcl 命令 `add_wave` 来完成相同操作，如下所示：

```
add_wave -into {hw_ila_data_1.wcfg} -radix hex { {counter1} }
```

在此示例中，在 `hw_ila_1` 的“Waveform Configuration”（波形配置）窗口中添加了探针 `counter1`，并且在“Waveform”窗口中其显示基数设为 `hex`。

使用波形 ILA 触发器和导出功能

图 160：**波形 ILA 触发器和导出功能**



X16755-032717

- “Enable Auto Re-Trigger”（启用自动重新触发）：选中“Waveform”（波形）窗口工具栏上的“Enable Auto Re-Trigger”（启用自动重新触发）按钮即可在成功完成触发 + 上传 + 显示操作后，启用 Vivado IDE 以自动重新装备与“Waveform”窗口触发器关联的 ILA 核。

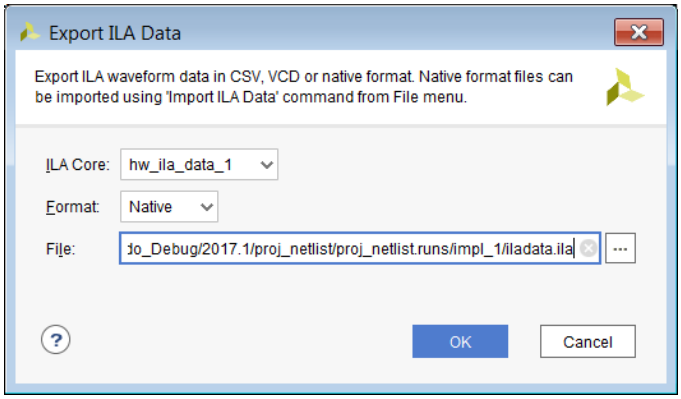
  每次成功完成触发事件后，对应于 ILA 核的“Waveform”窗口中显示的捕获数据都会被覆盖。“Auto Re-Trigger”（自动重新触发）选项可搭配“Run Trigger”（运行触发器）操作和“Run Trigger Immediate”（立即运行触发器）操作一起使用。单击“Stop Trigger”（停止触发器）按钮即可停止当前运行中的触发器。

- “Run Trigger”（运行触发器）：装备与“Waveform”窗口关联的 ILA 核，以检测由 ILA 核的基本或高级触发器设置所定义的触发器事件。

- “Run Trigger Immediate”（立即运行触发器）：装备与“Waveform”窗口关联的 ILA 核以忽略 ILA 核触发器设置，并立即触发该核。此命令用于通过捕获 ILA 核的探针输入处的任意活动来检测设计的“活动状态”。

- “Stop Trigger”（停止触发器）：停止与“Waveform”窗口关联的 ILA 的 ILA 核触发器。
- “Export ILA Data”（导出 ILA 数据）：从 ILA 核捕获数据并将其保存到文件。此数据可采用本机格式、`.csv` 或 `.vcd` 格式来捕获。在“Waveform”（波形）窗口工具栏上单击此图标后，会显示以下对话框。

图 161：**“Export ILA Data”对话框**



“ILA Core”（ILA 核）表示要为其导出数据的 ILA 调试核的名称。“Format”（格式）支持下列格式：Native（本机格式）、CSV 和 VCD。

- 本机格式可配置 `write_hw_ila_data` 命令，以默认 ILA 文件格式导出 ILA 数据，此文件可用于在其他时间重新导入 Vivado 以便您查看先前捕获的 ILA 数据。
- CSV 格式可配置 `write_hw_ila_data` 命令，按 `.csv` 文件格式导出 ILA 数据，此文件可用于将数据导入电子表格或第三方应用。
- VCD 文件格式可配置 `write_hw_ila_data` 命令，按 `.vcd` 文件格式导出 ILA 数据，此格式可用于导入第三方应用或查看器。

重要提示！虽然 ILA 数据可按 CSV、VCD 和本机 ILA 格式导出，但在 Vivado 中只能导入本机 ILA 格式。并且，仅支持将本机 ILA 数据导入 Vivado 用于脱机查看先前捕获的数据。探针信号不能用于其他目的，如触发等。

使用缩放功能

工具栏按钮便于快速使用波形缩放功能（请参阅下图）。或者，也可将鼠标滚轮与 Ctrl 键结合使用对当前选中波形进行缩放。

注释：缩放程度不是永久性的，并且会在 Vivado 会话之间进行复位。

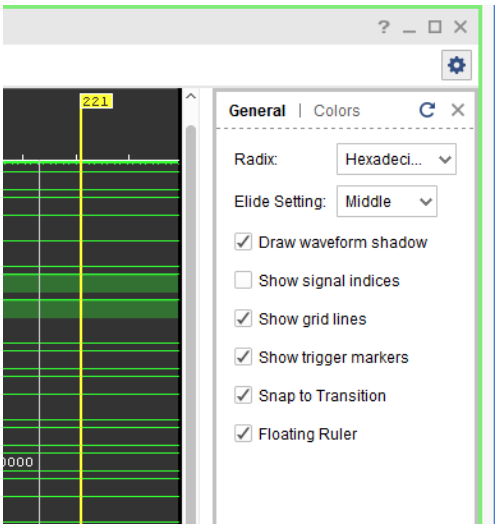
图 162：**波形缩放按钮**

Waveform Settings

波形查看器允许您自定义对象显示方式。

选中“Waveforms Settings”（波形设置）按钮时，将打开下图所示“Waveform Settings”窗口：

图 163：**Waveform Settings**



其中选项如下：

- “Colors”（颜色）选项卡：允许您选择波形对象的定制颜色
- “Radix”（基数）：设置总线探针的默认基数
- “Draw waveform shadow”（绘制波形阴影）：在标量“1”下显示亮绿色阴影以帮助区分“1”和“0”
- “Show signal indices”（显示信号索引）：在标量和总线名称左侧显示索引位置编号
- “Show trigger markers”（显示触发器标记）：在波形查看器中显示（或隐藏）红色触发器标记

自定义配置

您可使用下表中列示并简述的功能来自定义波形配置；其中功能名称链接至提供功能完整描述的相应小节。

表 27：**波形配置中的自定义功能**

功能特性	描述
光标	“Waveform”窗口中的主光标和辅助光标支持您显示并测量时间，两者构成了各种导航活动的焦点。
标记	您可添加标记以便浏览波形，并在特定时间显示波形值。
分频器	您可添加分频器以创建信号的可视化分隔符。

表 27：**波形配置中的自定义功能** (续)

功能特性	描述
组的使用	您可添加组，并在波形配置中将信号和总线添加到此类组中，这种方法可用于将一组相关信号组织在一起作为一个集合。
使用虚拟总线	您可向波形配置添加虚拟总线，以便将逻辑标量和阵列添加到其中。
重命名对象	您可重命名对象、信号、总线和组。
基数	默认基数用于控制波形配置、“Objects”（对象）面板和“Console”（控制台）面板中显示的总线基数。
总线位顺序	您可将总线位顺序从最高有效位 (MSB) 更改为最低有效位 (LSB)，反之亦然。

相关信息

光标
标记
分频器
组的使用
使用虚拟总线
重命名对象
基数
总线位顺序

光标

光标主要用作为样本位置的临时指示符并且会频繁移动，比如测量 2 个波形边沿之间的距离（以样本数为单位）时。

提示：要使用更为永久性的指示符（用于为多次测量确立时间基础等情况），请改为在“Waveform”（波形）窗口中添加标记。请参阅“标记”部分，以获取更多信息。

您只需在“Waveform”窗口中单击即可放置主光标。

要放置辅助光标，请按住 Ctrl 键 + 单击并按住波形，向左或向后拖动。您可在光标顶部看到 1 个标志用于标记位置。

或者，您也可以按住 Shift 键并单击波形中的任一点。这样主光标将留在原始位置，而在波形中您单击的位置处添加另一个光标。

注释：要保留辅助光标的位置同时定位主光标，请按住 Shift 键然后单击。通过拖动方式放置辅助光标时，必须拖动一段最短距离，然后才会显示辅助光标。

要移动光标，请悬停在光标上直至出现抓取符号，然后单击光标并将其拖至新位置。

在“Waveform”窗口中拖动光标时，如果选中“Snap to Transition”（对齐到转换位置）按钮（默认行为是选中此按钮），那么您会看到一个空心圆或实心圆。

- 空心圆 ○ 表示您当前位于选定信号的波形中的转换位置之间。
- 实心圆 ● 表示您正悬停在选定信号的波形转换上。通过单击“Waveform”窗口中没有任何光标、标记或浮动标尺的任意位置即可隐藏辅助光标。

相关信息

标记

标记

如果要以永久性方式来对波形内的重大事件进行标记，请使用标记。标记支持您测量与所标记的事件相关的距离（以样本数为单位）。

您可通过如下方式来添加、移动和删除标记：

- 您可在波形配置中主光标所在位置添加标记。
  1. 在“Waveform”（波形）窗口中，单击位于要添加标记处的样本编号处或者转换位置处即可在此处放置主光标。
  2. 选择“Edit”→“Markers”→“Add Marker”（编辑 > 标记 > 添加标记），或者单击“Add Marker”（添加标记）按钮。 

     这样即可在光标处放置标记，或者如果此光标位置处已存在标记，则会在稍偏移位置放置标记。该标记的样本编号会显示在行顶部。
- 您可通过拖放操作来将标记移至波形中其他位置。单击标记标签（位于标记顶部）并将其拖至相应的位置。
  - 拖动符号  表示该标记可移动。在“Waveform”窗口中拖动标记时，如果选中“Snap to Transition”（对齐到转换位置）按钮（默认行为是选中此按钮），那么您会看到一个空心圆或实心圆。
  - 实心圆  表示您当前正悬停在选定信号的波形转换处或者正悬停在另一个标记上。
  - 如果悬停在标记上，则实心圆为白色。
  - 空心圆  表示您当前位于选定信号的波形中的转换位置之间。
  - 松开鼠标以将标记拖到新的位置。
- 只需一条命令即可删除任一或全部标记。右键单击标记并执行以下任一操作：
  - 从弹出菜单中选中“Delete Marker”（删除标记）以删除单一标记。
  - 从弹出菜单中选中“Delete All Markers”（删除所有标记）以删除所有标记。

    注释：您也可以使用 Delete 键来删除选定的标记。
  - 使用“Edit”→“Undo”（编辑 > 撤销）可撤销标记删除操作。

触发器标记

红色触发器标记（其标签带有红色字母“T”）是特殊标记，表示在捕获缓冲器中出现触发事件。缓冲器中触发器标记的位置直接对应于“Trigger Position”（触发器位置）设置（请参阅“使用 ILA 默认仪表板”）。

注释：无法使用与常规标记相同的方法来移动这些触发器标记。请使用 ILA 核的“Trigger Position”属性设置来设置其位置。

相关信息

使用 ILA 默认仪表板

分频器

分频器用于在信号之间创建可视化分隔符。您可以将分频器添加到自己的波形配置中，以创建信号的可视化分隔符，如下所示：

1. 在“Waveform”（波形）窗口的“Name”（名称）列中，单击信号以在其下方添加分频器。
2. 在弹出菜单中，选择“Edit” → “New Divider”（编辑 > 新建分频器），或者右键单击并单击“New Divider”（新建分频器）。

此更改仅为视觉效果，不会向 HDL 代码添加任何内容。这样当您保存波形配置文件时，就会同时保存新的分频器。

您可按如下方式移动或删除分频器：

- 在波形中通过拖放分频器名称将分频器移至其他位置。
- 要删除分频器，请将其高亮，然后按 Delete 键，或者右键单击并从弹出菜单中单击“Delete”（删除）。

您也可以重命名分频器；请参阅“重命名对象”。

相关信息

重命名对象

组的使用

“Group”（组）是可展开和可折叠类别的集合，您可将波形配置中的信号和总线添加到组中以便将相关信号组合到一起。组本身不显示波形数据，但可展开以显示其中内容，也可折叠以隐藏内容。您可添加、更改和移除组。

要添加组，请执行以下操作：

1. 在波形配置中，选中 1 个或多个信号或总线以添加到组中。

   注释：每个组均可包含分频器、虚拟总线和其他组。
2. 选择“Edit” → “New Group”（编辑 > 新建组），或者右键单击并从弹出菜单中选择“New Group”（新建组）。

这样会将包含选定信号或总线的组添加到波形配置中。

每个组都会随“Group”按钮一并显示。 

此更改仅为视觉效果，不会向 ILA 核添加任何内容。

您可通过拖放信号或总线名称来将其他信号或总线移至组中。

您可按如下方式移动或移除组：

- 通过拖放组名来将组移至“Name”（名称）列中的其他位置。
- 要移除任一组，只需高亮组并依次选中“Edit” → “Wave Objects” → “Ungroup”（编辑 > 波形对象 > 取消分组），或者右键单击并从弹出菜单中选择“Ungroup”（取消分组）即可。这样会将原先位于该组中的信号或总线置于波形配置中的顶层层级内。

组也可以重命名；请参阅“重命名对象”。

注意！Delete 键可用于从波形配置中移除组及其嵌套的信号和总线。

相关信息

重命名对象

使用虚拟总线

您可向波形配置添加虚拟总线，虚拟总线是可供您添加逻辑标量和阵列的分组。虚拟总线可显示总线波形，其中在虚拟总线下按垂直顺序显示信号波形，并平铺为一维阵列。您可在添加虚拟总线后更改或移除这些虚拟总线。

要添加虚拟总线，请执行以下操作：

1. 在波形配置中，选中 1 个或多个信号或总线以添加到虚拟总线。
2. 选择“Edit”→“New Virtual Bus”（编辑 > 新建虚拟总线），或者右键单击并从弹出菜单中单击“New Virtual Bus”（新建虚拟总线）。

   虚拟总线会随“Virtual Bus”（虚拟总线）按钮  一起呈现。

   此更改仅为视觉效果，不会向 HDL 代码添加任何内容。

您可通过拖放信号或总线名称来将其他信号或总线移至虚拟总线中。保存波形配置文件时，就会保存新的虚拟总线及其嵌套的信号或总线。您也可以通过拖放虚拟总线名称来将其移至波形中的其他位置。

您可重命名虚拟总线；请参阅“重命名对象”。

要移除虚拟总线并取消其内容分组，请高亮该虚拟总线，并依次选中“Edit”→“Wave Objects”→“Ungroup”（编辑 > 波形对象 > 取消分组），或者右键单击并从弹出菜单中单击“Ungroup”（取消分组）。

注意！ Delete 键可用于从波形配置中移除虚拟总线及其嵌套的信号和总线。

相关信息

重命名对象

重命名对象

您可在“Waveform”（波形）窗口中重命名任意对象，例如，信号、分频器、组和虚拟总线。

1. 选中“Name”（名称）列中的对象名称。
2. 从弹出菜单中选中“Rename”（重命名）。
3. 将其名称替换为新名称。
4. 按 Enter 键或者单击名称范围外任意位置以使名称更改生效。

您可以双击对象名称，然后输入新名称。更改即时生效。波形配置中的对象名称更改不影响连接到 ILA 核探针输入的信号线的名称。

基数

了解总线上的数据类型至关重要。您需要明确基数设置与数据类型之间的关系才能有效使用“数字和模拟”的波形选项。请参阅“总线基数”，以了解有关基数设置及其对于“模拟波形”分析的影响的更多信息。

您可在“Waveform”（波形）窗口中更改个别信号（ILA 探针）的基数，如下所示：

1. 在“Waveform”（波形）窗口中右键单击总线。

2. 选中“Radix”（基数），然后从下拉菜单中选择所需格式：
   - 二进制
   - 十六进制
   - 无符号十进制
   - 有符号十进制
   - 八进制

   重要提示！在“Objects”（对象）窗口中对任一项的基数进行的更改并不会应用于“Waveform”窗口或 Tcl 控制台中的值。要更改“Waveform”窗口中的任一信号（ILA 探针）的基数，请使用“Waveform”窗口弹出菜单。

   - 真实情况下，最大总线宽度为 64 位。如果总线宽度超过 64 位，则可能出现错误的值。
   - 浮点仅支持 32 位和 64 位阵列。

相关信息

总线基数

使用浮动标尺

浮动标尺使用样本数值库来辅助样本测量，取代使用“Waveform”窗口顶部的标准标尺上所示的绝对样本数值。

您可显示（或隐藏）浮动标尺，也可将其移至“Waveform”窗口中的任一位置。浮动标尺的样本库（样本 0）属于辅助光标，或者如果没有辅助光标，则作为选定的标记来使用。

仅当辅助光标（或选定标记）存在时，浮动标尺按钮和浮动标尺本身才可见。

1. 请执行以下任一操作以显示或隐藏浮动标尺：
   - 放置辅助光标。
   - 选择标记。
2. 选择“View” → “Floating Ruler”（视图 > 浮动标尺）。

   首次执行此操作时，只需遵循该过程即可。每次放置辅助光标或选择标记时，都会显示浮动标尺。

   再次选择该命令即可隐藏浮动标尺。

总线位顺序

您可在波形配置中反转总线位顺序以在 MSB 优先和 LSB 优先信号表示法之间进行切换。

要反转位顺序，请执行以下操作：

1. 选择总线。
2. 右键单击并选中“Reverse Bit Order”（反转位顺序）。

这样即可反转总线位顺序。“Reverse Bit Order”命令会标记为显示这是当前行为。

总线基数

总线值解读为数值，由总线波形对象上的基数设置来确定，如下所示：

- 使用二进制、八进制、十六进制、ASCII 和无符号十进制基数会导致将总线值解读为无符号整数。总线上的数据格式必须与基数设置相匹配。
- 任何非 0 或非 1 位都会导致将整个值解读为 0。
- 有符号十进制基数则会导致将总线值解读为有符号整数。

查看模拟波形

要将数字波形转换为模拟波形，请执行以下操作：

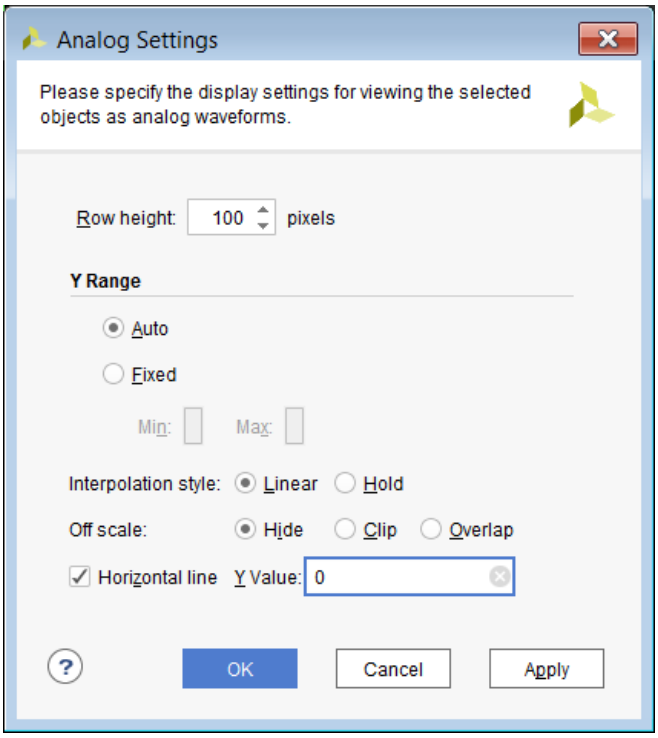
1. 在“Waveform”（波形）窗口的“Name”（名称）区域中，右键单击总线。
2. 选择“Waveform Style”（波形样式）和“Analog Settings”（模拟设置）以选中相应的绘制设置。

这样即可将总线的数字波形图转换为模拟格式。

您可通过选中并拖拽各行来调整模拟波形或数字波形的高度。

下图所示“Analog Settings”对话框包含对应模拟波形图的设置。

图 164： **“Analog Settings”对话框**



“Analog Settings”对话框选项如下：

- “Row Height”（行高）：指定所选波形对象的高度（以像素为单位）。更改行高不会影响垂直方向显示或隐藏的波形，而只是对波形高度进行伸缩。

  在“模拟波形”与“数字波形”之间进行切换时，行高会设置为对应波形的相应默认值（针对数字波形为 20，针对模拟波形为 100）。

- “Y Range”（Y 范围）：指定波形区域内显示的数字值范围。
  - “Auto”（自动）：指定只要发现窗口内可见时间范围超出当前范围，此范围就持续扩展。
  - “Fixed”（固定）：指定时间范围保持固定间隔不变。
  - “Min”（最小值）：指定波形区域底部显示的值。
  - “Max”（最大值）：指定波形区域顶部显示的值。

  这两个值均可指定为浮点值，但如果波形对象基数为整数，则这些值将被截位至整数。

- “Interpolation Style”（内插样式）：指定用于连接数据点的线的绘制方式。
  - “Linear”（线性）：指定 2 个数据点之间为直线。
  - “Hold”（保持）：指定在 2 个数据点之间，从左侧点绘制 1 条水平线到右侧点的 X 坐标，然后绘制另一条线以将前一条线连接到右侧数据点，构成 L 形。
  - “Off Scale”（超标度）：指定超出波形区域的 Y 范围的波形值的绘制方式。
  - “Hide”（隐藏）：指定不显示超出范围的值，例如，达到波形区域上限或下限的波形将消失，直至值重新恢复到范围内为止。
  - “Clip”（剪切）：指定更改超出范围的值，使其位于波形区域顶部或底部，这样达到波形区域的上限或下限的波形就会沿边界呈现为水平线，直至值重新恢复到范围内为止。
  - “Overlap”（重叠）：指定在波形值达到波形窗口本身上限的任意位置都需绘制波形，即使其值在波形区域边界外并与其他波形重叠。

- “Horizontal Line”（水平线）：指定是否在给定值绘制水平规则。如果开启此复选框，则会在指定 Y 值的垂直位置绘制 1 条水平网格线，前提是该值在波形的 Y 范围内。

  就像“Min”和“Max”一样，Y 值接受浮点值，但如果所选波形对象的基数为整数，则会被截位为整数。

重要提示！模拟设置保存在波形配置中；但由于 Y 维度内缩放控制具有高度交互性（不同于基数等其他波形对象属性），因此不影响波形配置的修改状态。因此，缩放设置不随波形配置一起保存。

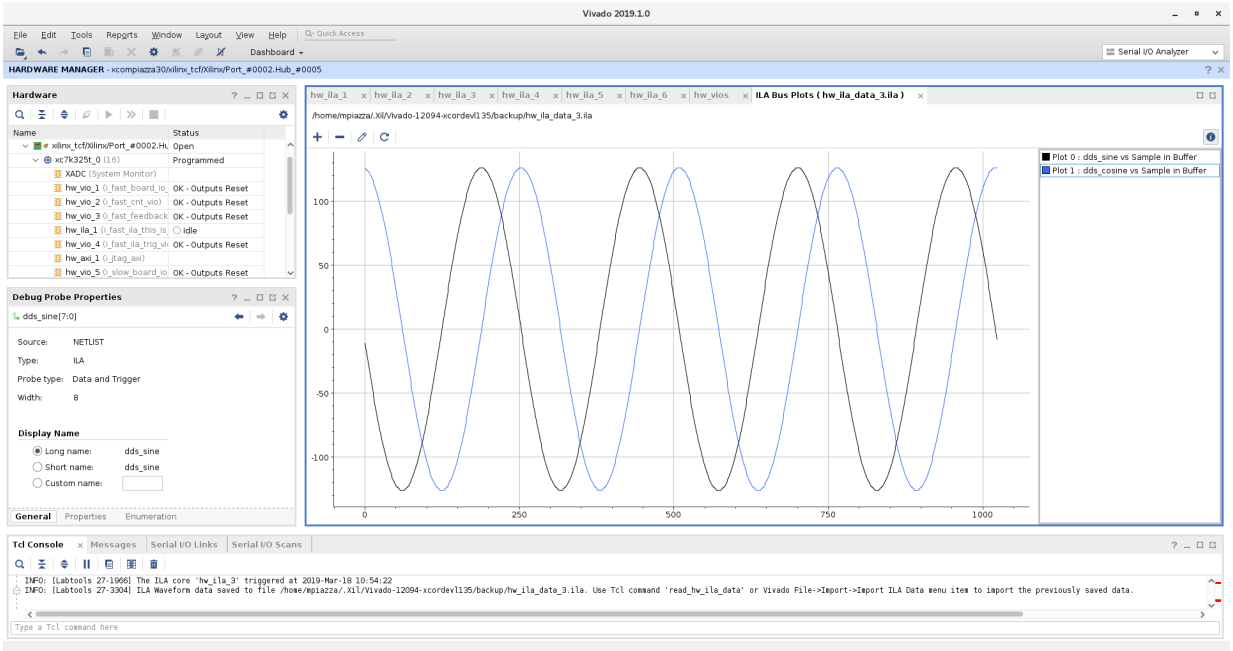
总线图查看器

除模拟波形查看器外，Vivado 硬件管理器还支持 Bus Plot Viewer（总线图查看器），此查看器允许查看总线值随时间的变化，或者在 X 轴对比 Y 轴上所绘制的 2 个不同总线值的比较。

如需执行以下任一操作，则查看“Bus Plot”（总线图）很有用：

- 绘制模拟样本数据与时间的对比
- 绘制模拟样本数据与模拟样本数据的对比

图 165：**显示缓冲器中触发数据与样本对比的总线图**

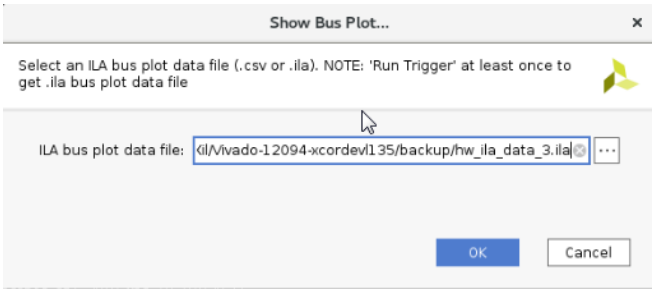


创建总线图

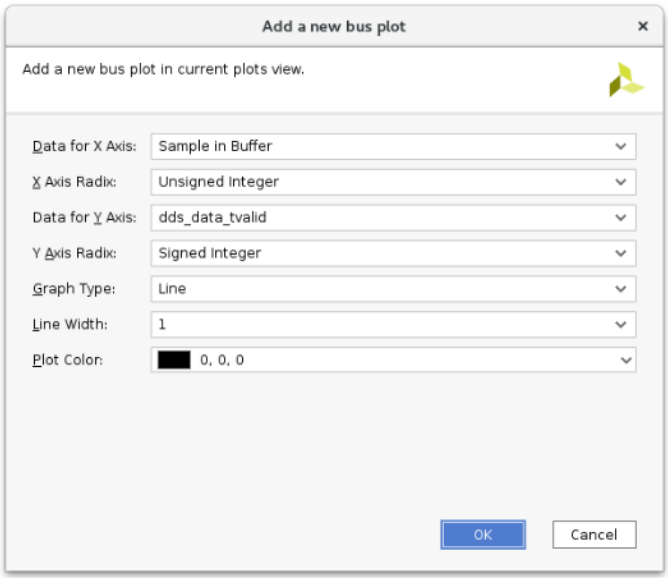
“Bus Plot”（总线图）查看器可用于绘制先前捕获的 ILA 走线数据。因此，它需要指向 ILA 总线图数据文件（`.csv` 或 `.ila`）的路径。默认情况下，“Show Bus Plot”（显示总线图）对话框会选择最近自动保存的 ILA 触发数据。

总线图创建示例

1. 要创建总线图，请打开 Vivado 硬件管理器并选中“Tools” → “Show Bus Plot”（工具 > 显示总线图）。
2. 这样会显示“Show Bus Plot”对话框，以便您选择先前保存的 ILA 数据文件。默认情况下，会自动选中对应于最近一次 ILA 追踪的自动保存的 ILA 数据文件。要选择先前保存的 ILA 数据文件，请输入指向对应 `.ila` 或 `.csv` 文件的路径。



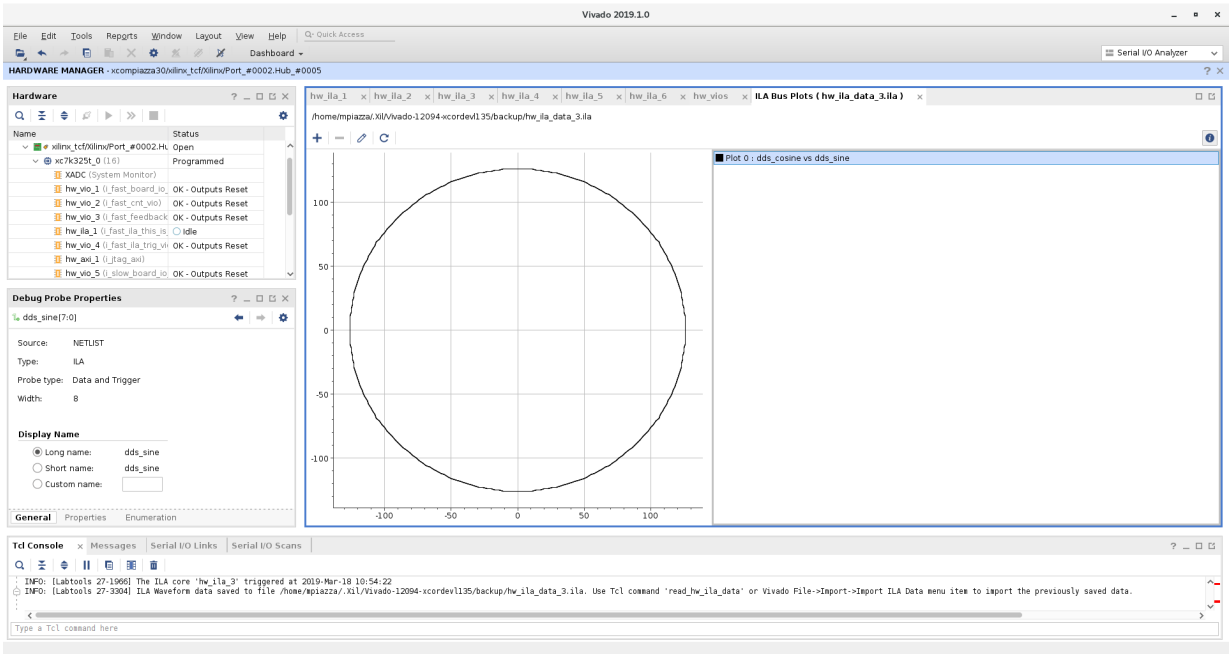
3. 单击“OK”（确定），这样会显示空白“Bus Plot”（总线图）窗口。要添加总线图，请单击“+”符号，这样会显示 1 个对话框，其中包含配置新总线图的选项。



“Add New Bus Plot”（添加新总线图）选项如下所示：

- “Data for X Axis”（X 轴数据）：指定用于 X 轴的总线数据。
  - “Sample in Buffer”（缓冲器中的样本数）：ILA 捕获缓冲器中的 ILA 样本数。
  - “Sample in Window”（窗口中的样本数）：捕获窗口中的 ILA 样本数。如果选中任一捕获窗口，那么此数值与缓冲器中的样本数相同，但如果使用多个捕获窗口，那么此数值表示给定捕获窗口中的样本数。
  - “TRIGGER”（触发器）：捕获窗口中的触发器位置。
- “X Axis Radix”（X 轴基数）：指定绘制 X 轴数据时要使用的基数。
  - “Signed Integer”（有符号整数）。
  - “Unsigned Integer”（无符号整数）。
- “Data for Y Axis”（Y 轴数据）：指定用于 Y 轴的总线数据。
  - “Sample in Buffer”（缓冲器中的样本数）：ILA 捕获缓冲器中的 ILA 样本数。
  - “Sample in Window”（窗口中的样本数）：捕获窗口中的 ILA 样本数。如果选中任一捕获窗口，那么此数值与缓冲器中的样本数相同，但如果使用多个捕获窗口，那么此数值表示给定捕获窗口中的样本数。
  - “TRIGGER”（触发器）：捕获窗口中的触发器位置。
- “Y Axis Radix”（Y 轴基数）：指定绘制 Y 轴数据时要使用的基数。
  - “Signed Integer”（有符号整数）。
  - “Unsigned Integer”（无符号整数）。
- “Graph Type”（计算图类型）
  - “Line”（线图）：将总线图显示为连接离散样本的连续线段。
  - “Point”（点图）：将总线图显示为表示离散样本的点。
- “Line Width”（线宽）：指定用于在总线图查看器中绘制信号的宽度。
- “Plot Color”（图色）：允许选择不同颜色以绘制总线图。

4. 配置总线图后，单击“OK”以将总线图添加到 Vivado 硬件管理器中。这样即可显示总线图。

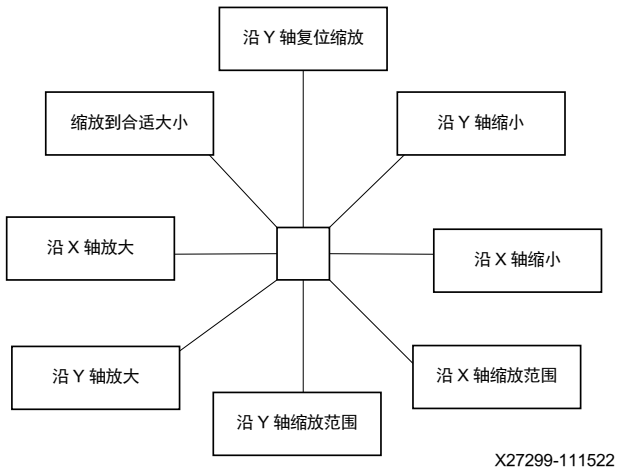


注释：退出 Vivado 硬件管理器时，不保存总线图设置。关闭 Vivado IDE 前，请确保所需测量已全部完成。

缩放手势

除了支持在 X 维进行缩放的缩放手势外，在模拟波形上还可使用额外的缩放手势，如下图所示。

图 166：**模拟缩放选项**



要调用缩放手势，请按住鼠标左键并按图中所示方向拖动，其中起始鼠标位置即图中的中心点。

其他缩放手势如下所示：

- “Zoom Out Y”（沿 Y 轴缩小）：沿 Y 维度按 2 次幂缩小，幅度由释放鼠标按键的位置与起点的距离决定。执行缩放时，起始鼠标位置的 Y 值保持静止。

- “Zoom Y Range”（沿 Y 轴缩放范围）：绘制垂直幕布，以指定松开鼠标时要显示的 Y 范围。
- “Zoom In Y”（沿 Y 轴放大）：沿 Y 维度按 2 次幂放大，幅度由释放鼠标按键的位置与起点的距离决定。

  执行缩放时，起始鼠标位置的 Y 值保持静止。
- “Reset Zoom Y”（沿 Y 轴复位缩放）：将 Y 范围复位至波形窗口中当前显示的值对应的 Y 范围，并将 Y 范围模式设置为“Auto”（自动）。

Y 维度的所有缩放手势都可设置 Y 范围模拟设置。“Reset Zoom Y”可将“Y Range”设置为“Auto”，其他手势可将“Y Range”设置为“Fixed”（固定）。

第 15 章

实现后的设计调试

您可以在实现后修改、添加或删除调试核。在 AMD Vivado™ Design Suite 中提供了 2 种办法来完成此类操作。

如果要替换与 ILA 核的现有连接，AMD 建议您使用 ECO 流程。ECO 流程可对已实现的检查点 (DCP) 进行操作，并且可以节省原本用于对设计进行完整重新布线的时间。

如果要添加新的 ILA 核、删除现有 ILA 核或者修改现有 ILA 核（例如，调整探针宽度大小、更改数据深度等），AMD 建议您使用增量编译流程。适用于调试核的“增量编译”流程在已综合的设计或检查点 (DCP) 上运行并使用已实现的参考检查点，最好是来自先前运行的实现的检查点。这样即可节省原本用于对设计进行完整重新布线的时间。

以下章节详细讲解了上述每一种调试相关流程。

使用 Vivado ECO 流程来替换现有调试核

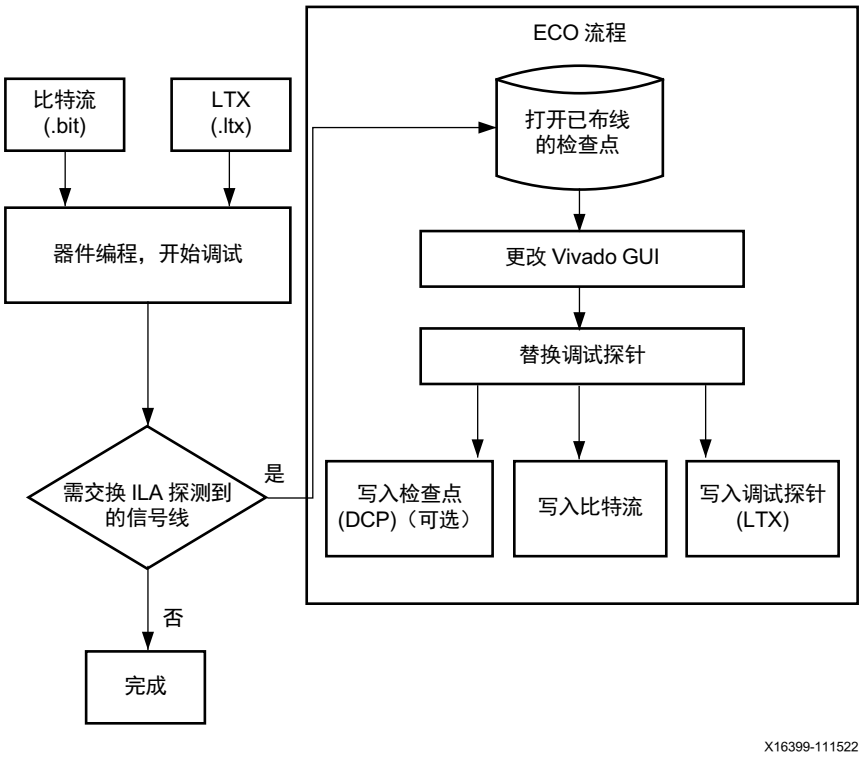
在已布局布线的设计检查点中可以替换已连接到 ILA 核的调试信号线。您可使用“Engineering Change Order (ECO)”（工程变更单 (ECO)）流程来执行此操作。这是高级设计流程，用于接近完成的设计，在此流程中您需要交换已连接到 ILA 探针端口的信号线。此方法的用途主要有二：

- **节省时间。**该功能允许您将正在探测的现有调试信号线交换为其他信号线。
- **它侵入性最低。**替换所探测的信号线后，需要将这些信号线布线到调试核的输入。设计其余部分保持不变，因此不仅保留了原先的实现结果，还可以使您正尝试发现的错误尽可能免于因重新实现而消失。

> **重要提示！**此流程仅适用于已在其中例化或插入 ILA 核的设计。

下图显示了使用 ECO 设计流程来替换调试信号线的进程。

图 167：**调试 ECO 设计流程**

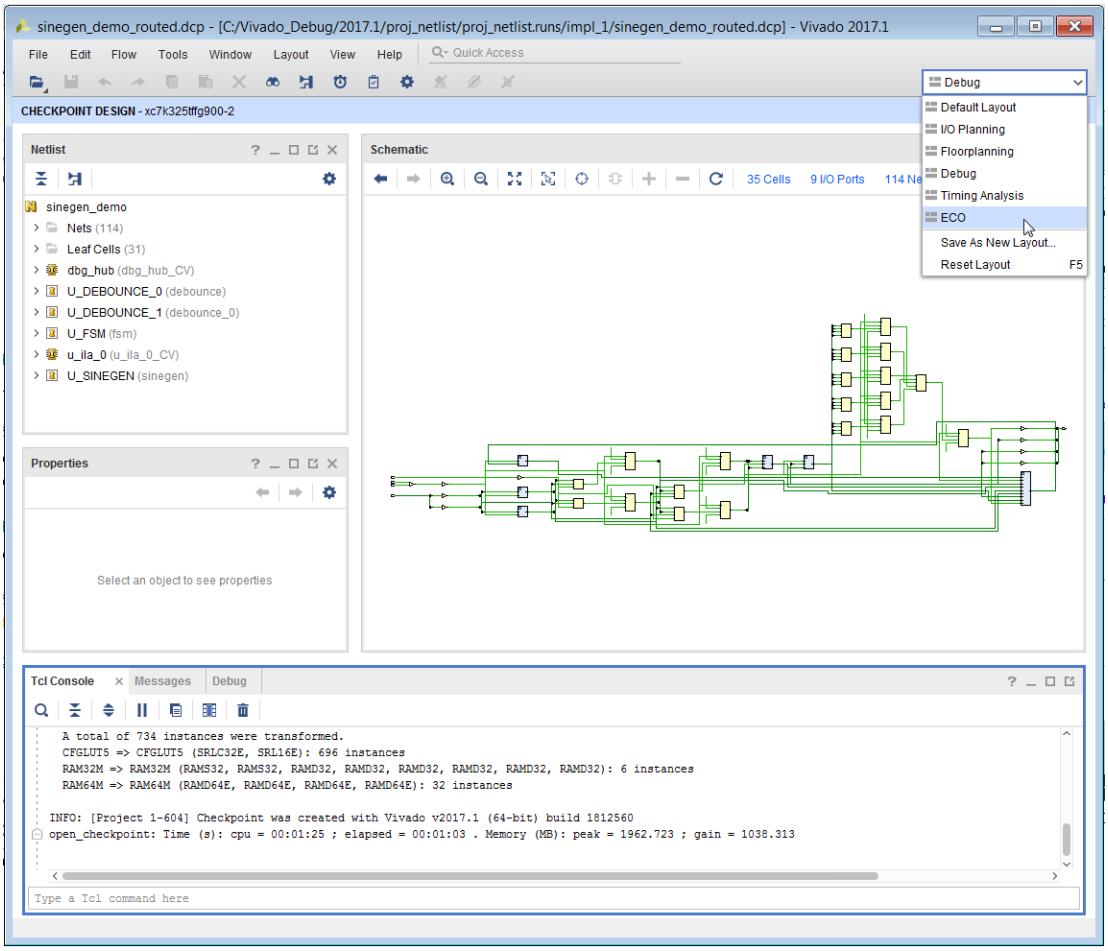


在已布局布线的设计检查点上替换调试探针

使用 Vivado 硬件管理器来对器件上已完成烧录的设计进行调试时，有时需要将正在进行调试探测的信号线替换为其他信号线。在此情况下，您无需重新更改 RTL 代码或更改已插入的调试核中正在探测的信号线，而可改用 ECO 流程来替换调试信号线。

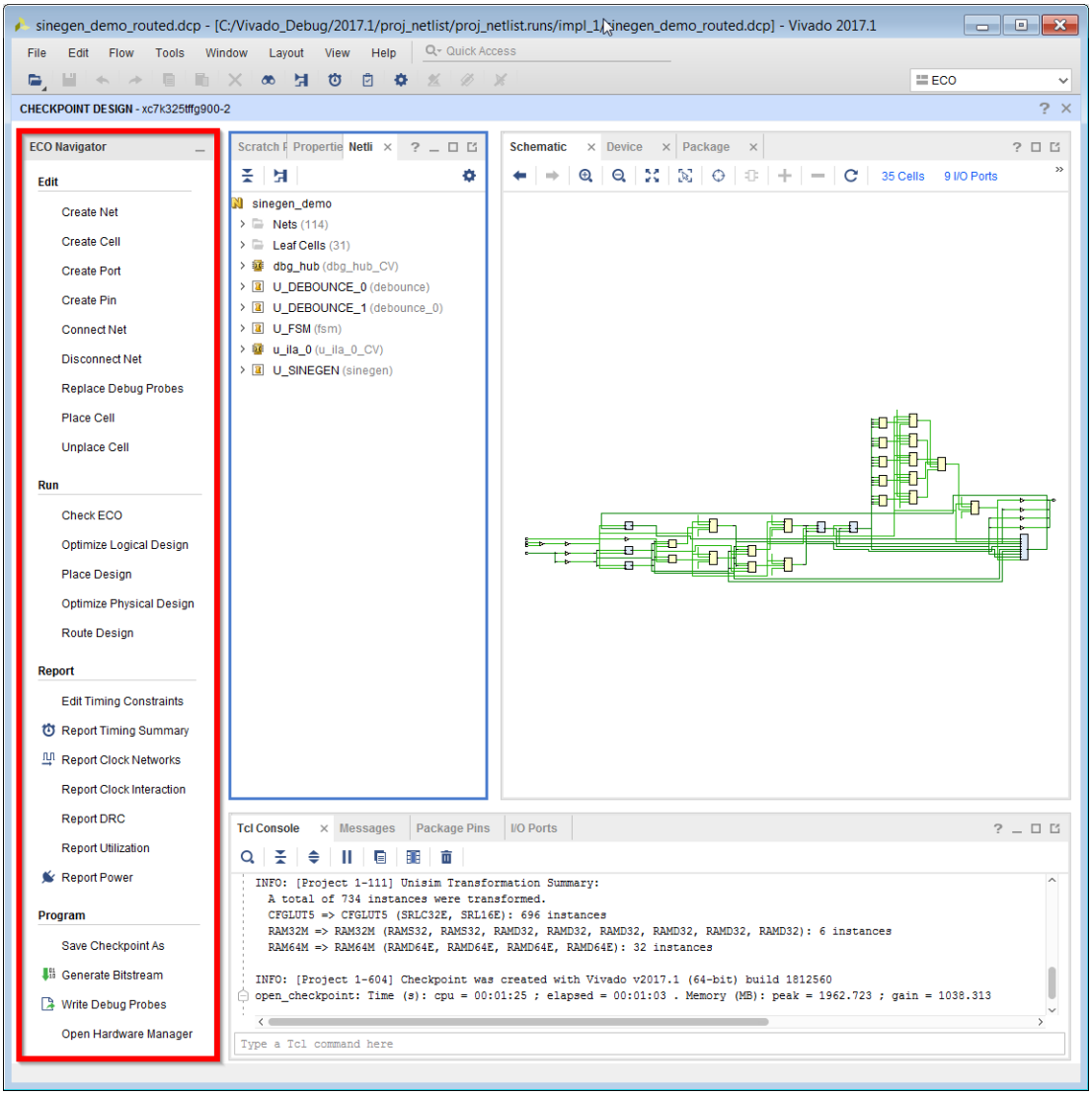
要使用 ECO 流程，请在 Vivado IDE 中打开已布局布线的设计检查点 (DCP)，并将布局切换为 ECO。

图 168：**选择 ECO 布局**



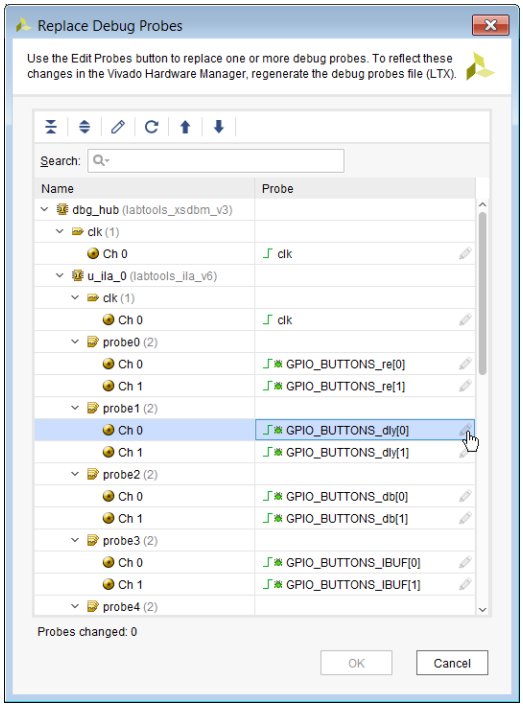
这样 Flow Navigator 就会切换至包含一组不同选项的 ECO Navigator。

图 169：**ECO Navigator**



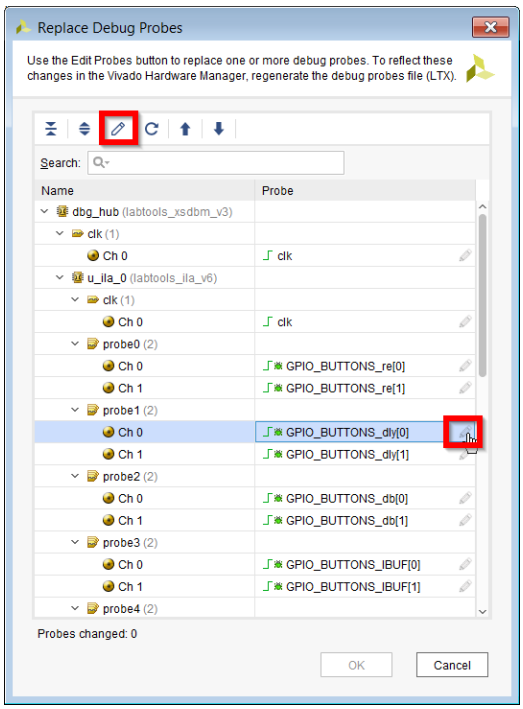
在 ECO Navigator 中，单击“Replace Debug Probes”（替换调试探针），这样会开启“Replace Debug Probes”对话框。

图 170：**“Replace Debug Probes”对话框**



在“Replace Debug Probes”对话框中，高亮要更改其信号线的探针，然后单击“Edit Probes”（编辑探针）按钮。使用每个探针右侧的“Edit Probes”按钮即可更改各信号线。或者，也可使用窗口左侧边缘处的“Edit Probes”按钮来更改多个探针的信号线。

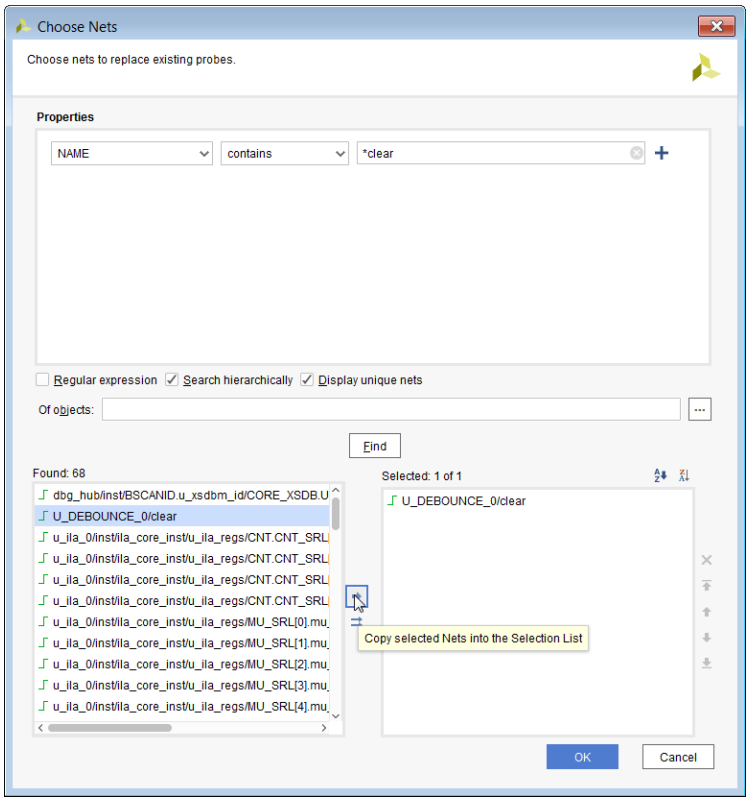
图 171：**“Edit Probes”按钮**



单击“Edit Probes”按钮以开启“Choose Nets”（选择信号线）对话框，您可在其中选择信号线以便用于替换现有信号线。

输入查找条件以便选择要用于替换现有信号线的信号线。如果查找条件返回的信号线数量超过 10000，则请优化查找条件，然后重试。在左侧查找结果上选择所需的信号线，然后单击箭头（“->”）以将这些信号线添加到右侧“Selected”（选定的）信号线名称列中。请确保右侧“Selected”列中的信号线与要替换的信号线数量相匹配。单击“OK”（确定）以继续。

图 172：**“Choose Nets”对话框**



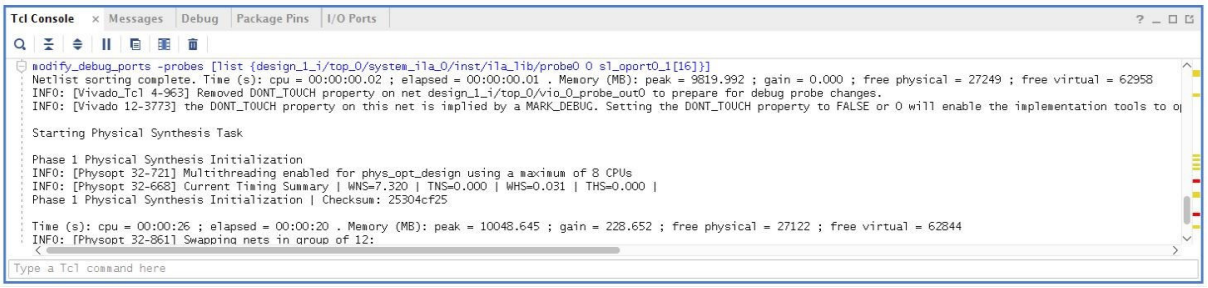
重要提示！完成替换所有必要调试探针后，请对其进行重新布线，并重新生成比特流，您必须重新生成调试探针文件 (`.ltx`)。

提示：您还可通过单击“Replace Debug Probes”对话框中左侧的“Edit Probes”按钮来选择多条信号线或单一总线。

替换调试核上的所有目标信号线后，请单击“OK”以开启确认对话框，以便您确认要执行的更改。

重要提示！检查 Tcl 控制台，确保其中没有任何“Warnings/Errors”（警告/错误）。

图 173：**修改调试探针 Tcl 消息**



删除要探测的信号线路径上的任意信号线段都可能影响硬件管理器中显示的探针名称。Vivado IDE 会选择距离所探测的信号线最近并含有 MARK_DEBUG 属性的信号线段。如果找不到含 MARK_DEBUG 属性的信号线段，则会选择顶层信号线。如有多个含 MARK_DEBUG 属性的信号线段，则该工具会随机选择其中之一。

替换所有调试探针端口后，您可使用 ECO Navigator 中的“Save Checkpoint As”（检查点另存为）选项来将所做的修改保存到新的检查点。您需要运行 ECO Navigator 中的“Replace Debug Probes”命令以便为调试探针生成新的 `.ltx` 文件。您应生成新的比特文件以便烧录器件。您可连接至 Vivado 硬件管理器，以便对含新更改的设计进行调试。

用于替换现有调试探针的 Vivado ECO Tcl 流程

您可以使用 Vivado Tcl 流程替代先前章节中所述的 GUI 流程。以下 Tcl 命令可用于修改调试核所探测的信号线。

```
modify_debug_ports -probes [list {top/x_ila/probe0 0 top/inst_A/net_0} \
      {top/x_ila/probe1 1 top/inst_A/net_a} {top/x_ila/probe1 2 top/inst_A/
net_b}]
```

此命令可执行所有网表修改，以断开现有信号线与指定探针端口之间的连接。在此示例中，现有信号线与 ILA 的索引 0 上的探测端口 0 以及索引 1 和索引 2 上的探测端口 1 之间的连接全部断开。其中每个探针都分别连接到指定为 `net_0`、`net_a` 和 `net_b` 的信号线。修改后的连接也会自动完成布线。此进程中断开连接的信号线仍保留处于断开状态。

通过修改调试核 (ILA) 来进行增量编译

增量编译是高级设计流程，用于接近完成且需要少量更改的设计。重新综合这些少量更改后，流程将能够：

- 缩短布局布线运行时间。
- 保留 QoR 可预测性，因为它复用先前来自参考设计的布局布线。当综合更改与参考设计的相似性达到至少 95% 时，此流程最有效。

通过使用“增量编译”设计流程来重新实现设计，即可将增量调试更改应用于已布局布线的设计。在以下情况下建议使用此流程：

- 已实现的现有设计中无调试核，或者
- 您需要通过更改探针宽度、数据深度等内容来修改现有调试核。
- 您需要从设计中删除调试核。

增量编译流程设计

“Incremental Compile”（增量编译）流程涉及 2 种不同设计：参考设计与含调试核修改的当前设计。

参考设计

参考设计通常为已完成综合和布局布线的当前设计的早期迭代或变体。但您可使用检查点，且不限其中所含布局和/或布线数量。参考设计检查点 (DCP) 可以是大量设计迭代的产物，其中涉及达成时序收敛所需的代码变更、布局规划和约束修改。加载完当前设计后，使用 `read_checkpoint -incremental <dcp>` 命令即可加载参考设计检查点。使用 `-incremental` 选项来加载参考设计检查点即可支持增量编译设计流程，以便后续执行布局布线操作。

当前设计

当前设计相比参考设计，具有少量调试相关的设计更改或变动。这些更改或变动可包括：

- 调试核 RTL 例化更改
- 调试核插入更改
- 调试核相关 RTL 更改和插入更改

要在已实现的现有设计中插入、删除或修改调试核，请打开已综合的 DCP 或设计，然后使用调试插入流程。在“使用网表插入调试探测流程”中可找到有关调试插入流程的详细信息。

您也可以修改现有调试核，或者将新的调试核例化到现有 RTL 设计中。“增量编译”流程复用来自参考设计的布局布线并附带新的调试相关修改。在“HDL 例化调试探测流程概述”中可找到有关调试例化流程的详细信息。

相关信息

使用网表插入调试探测流程
HDL 例化调试探测流程概述

使用增量编译

在工程模式和非工程模式下，使用 `read_checkpoint -incremental <reference_dcp_file>` 命令加载参考设计检查点时会进入增量布局布线模式，此命令中，`<reference_dcp_file>` 用于指定参考设计检查点的路径和文件名。使用 `-incremental` 选项来加载参考设计检查点即可支持增量编译设计流程，以便后续执行布局布线操作。在非工程模式下，`read_checkpoint -incremental` 的执行应：(1) 晚于 `opt_design` 且 (2) 早于 `place_design`。如果使用调试插入流程，那么调试核相关的 XDC 命令应先于 `opt_design` 执行。

在非工程模式下使用增量编译

要在非工程模式下指定使用某个设计检查点文件 (DCP) 作为参考设计并运行增量布局，请执行以下操作：

1. 打开已综合的 DCP，加载已综合的设计。
2. 运行调试核命令。
3. 运行 `opt_design`。

   重要提示！确保 `opt_design` 选项和指令与原始参考运行中所使用的选项和指令尽可能匹配。

4. 运行 `read_checkpoint -incremental <reference_dcp_file>`。

5. 运行 `place_design`。
6. 运行 `route_design`。

```
# to load the current design
link_design;
#Create the debug core
create_debug_core u_ila_0 ila
#set debug core properties
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
#connect the probe ports in the debug core to the signals being probed
in the design
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
create_debug_port u_ila_0 probe
opt_design
read_checkpoint -incremental <reference_dcp_file>
place_design
route_design
```

重要提示！您必须打开已综合的检查点才能修改设计中的调试核。不支持通过打开布线后检查点来插入调试核。

在工程模式下使用增量编译

在工程模式下，您可在“Design Runs”（设计运行）窗口中设置增量编译选项。

要设置增量编译选项，请执行以下操作：

1. 在“Design Runs”窗口中，选择运行。
2. 单击上下文菜单中的“Set Incremental Compile”（设置增量编译）。
3. 在“Set Incremental Compile”窗口中，选择参考设计检查点。这样即可启用并运行增量编译模式。
4. 打开综合后网表，在其中可选择修改或添加 RTL 中已例化的调试核。
5. 使用“Set Up Debug”Wizard（设置调试向导）来对已插入设计的调试核执行插入、删除或修改操作。
6. 实现设计。

重要提示！您必须打开已综合的设计才能修改设计中的调试核。不支持通过打开布线后设计来插入调试核。

如需了解更多信息，请参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的增量编译功能特性。

检验参考设计与当前设计之间的相似性

运行 `report_incremental_reuse` 以检验并报告参考设计检查点文件与当前设计之间的相似性。`report_incremental_reuse` 命令可将来自参考设计检查点的网表与当前存储器内的设计进行比较并报告单元、信号线和端口的匹配百分比。

设计相似度越高，则复用来自参考设计的布局布线的效率越高。参考设计与当前设计之间相似性百分比越高，则布局布线复用机会越大。

第 16 章

串行 I/O 硬件调试流程

相关信息

在硬件中调试串行 I/O 设计

串行 I/O 硬件调试流程

注释：如需了解 AMD Versal™ 串行 I/O 硬件调试流程，请参阅 第 17 章：Versal 串行 I/O 硬件调试流程。

AMD Vivado™ IDE 提供了一条生成设计的捷径，以帮助您对使用 AMD 高速千兆位收发器 (GT) 技术的系统进行调试和验证。系统内串行 I/O 调试流程包含 3 个不同阶段：

1. IBERT 核生成阶段：自定义并生成适合满足您的硬件高速串行 I/O 要求的 IBERT 核。
2. IBERT 设计示例生成和实现阶段：为上一步生成的 IBERT 核生成设计示例。
3. 串行 I/O 分析阶段：与设计中包含的 IBERT IP 交互，对高速串行 I/O 链路中的问题进行调试和验证。

本章其余部分演示了如何完成前 2 个阶段。“在硬件中调试串行 I/O 设计”中涵盖了第三阶段的相关内容。

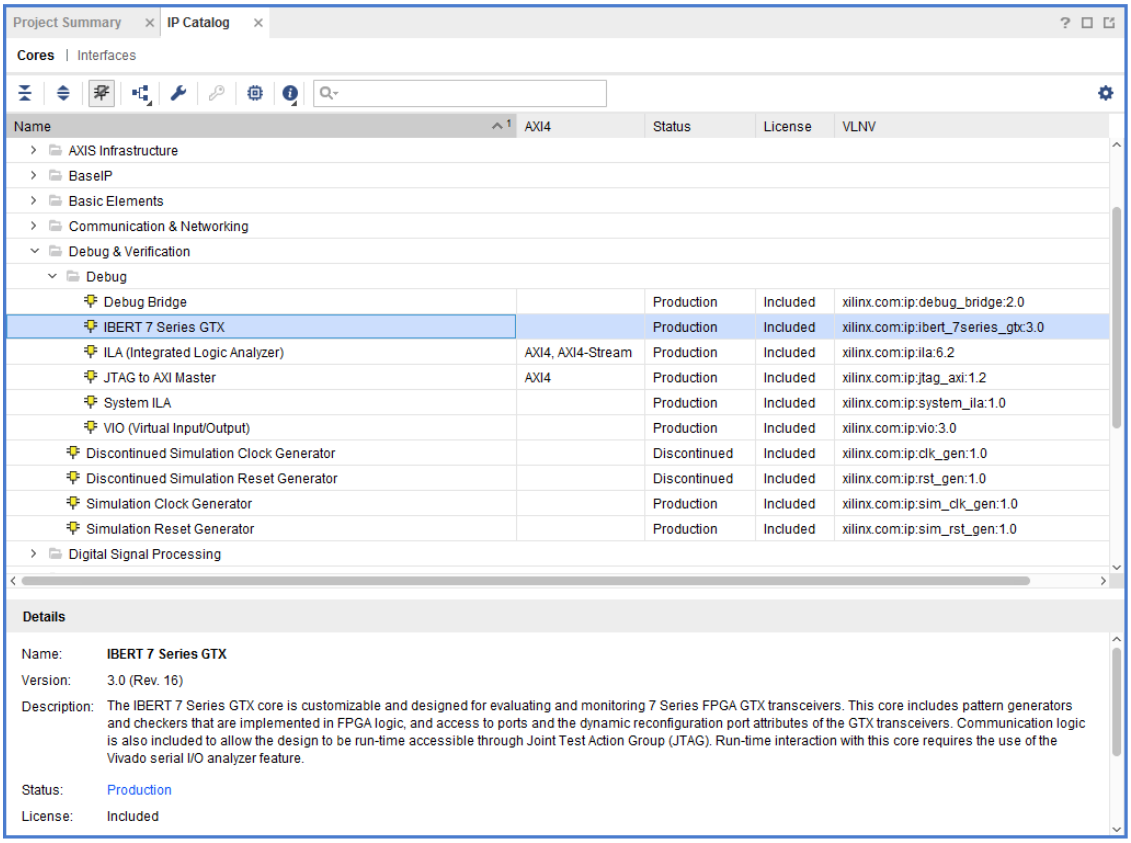
使用 Vivado IP 目录生成 IBERT 核

要获取适合的硬件设计以帮助完成系统高速串行 I/O 接口的调试和确认，首先要做的是生成 IBERT 核。详细步骤总结如下：

1. 打开 Vivado IDE。
2. 在首个面板上选择“Manage IP”→“New IP Location”（管理 IP > 新建 IP 位置），在“Open IP Catalog” Wizard（打开 IP 目录向导）打开后，单击“Next”（下一步）。
3. 选择期望的部件、目标语言、目标仿真器以及 IP 位置。单击“Finish”（完成）。
4. 根据上一步中所选的器件，在“IP Catalog”（IP 目录）的“Debug and Verification”→“Debug”（调试和验证 > 调试）下可找到 1 个或多个可用的 IBERT 核，如下图所示。
5. 双击所需 IBERT 架构以打开对应该核的“Customize IP”Wizard（自定义 IP 向导）。

按给定硬件系统要求，自定义该 IBERT 核。如需获取有关各可用 IBERT 核的详细信息，请参阅下列 IP 文档：

- 《Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP 产品指南》(PG132)
- 《Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP 产品指南》(PG133)
- 《Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP 产品指南》(PG152)



生成并实现 IBERT 设计示例

生成 IBERT IP 核后，它会在“Sources”窗口中显示为 `ibert_7series_gtx` 或其他类似名称。要生成设计示例，请在“Sources”（源）窗口中右键单击此 IBERT IP，单击“Open IP Example Design”（打开 IP 设计示例），然后在出现的对话框窗口中指定设计工程示例的目标位置。此命令会为设计示例打开新的 Vivado 工程窗口，并向该工程添加相应的顶层封装文件和约束，如下图所示。

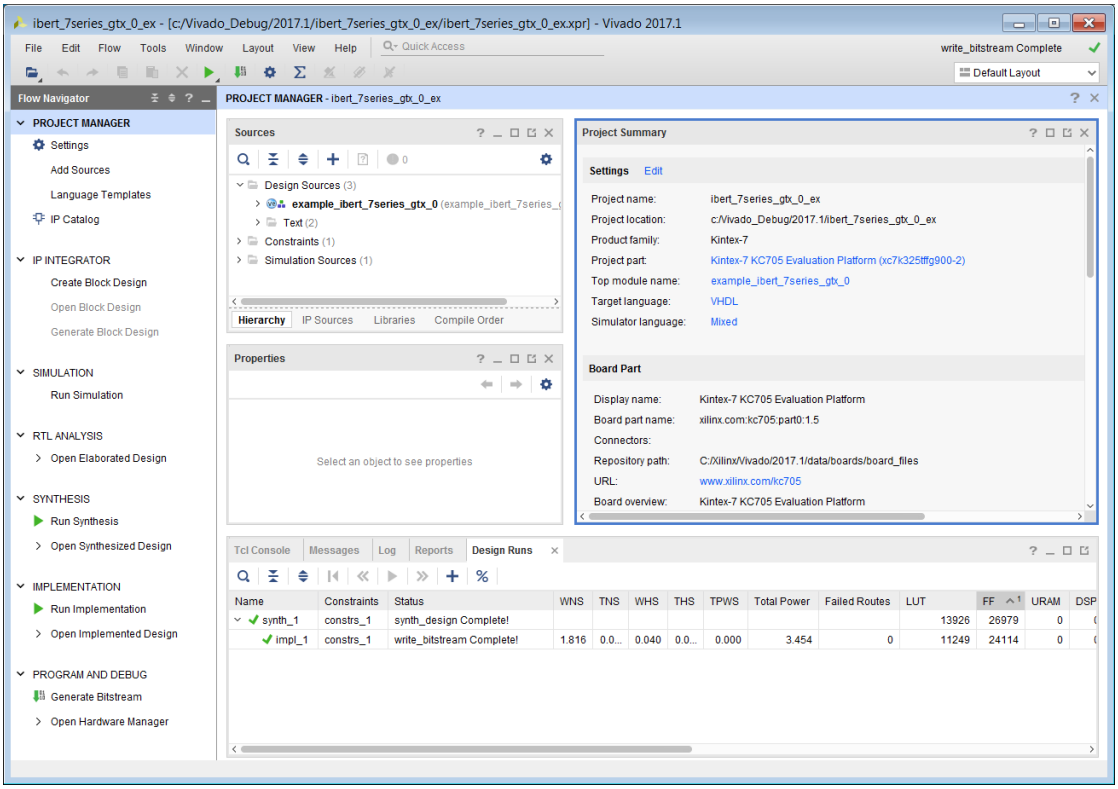
重要提示！建议不要对 IBERT IP 设计示例进行修改，因为修改可能导致在硬件中与 IBERT IP 核进行交互时发生功能问题。

生成设计示例后，即可在 Vivado IDE Flow Navigator 的“Program and Debug”（烧录和调试）部分中单击“Generate Bitstream”（生成比特流），或者运行以下 Tcl 命令，通过比特流创建核来实现 IBERT 设计示例：

```
launch_runs impl_1 -to_step write_bitstream wait_on_run impl_1
```

如需了解有关各种设计实现方法的更多详细信息，请参阅《Vivado Design Suite 用户指南：设计流程概述》(UG892)。

图 174：**IBERT 设计示例**



In-System IBERT 系统串行 I/O 设计调试流程

重要提示！In-System IBERT 核仅可用于 UltraScale 和 UltraScale+ 器件系列，在 Versal 器件系列上不受支持，因为 In-System IBERT 功能已集成到 Versal IBERT 中。

In-System IBERT IP 允许您使用 Vivado Serial IO Analyzer 工具对 UltraScale 和 UltraScale+ 收发器执行 2D 眼图扫描。当收发器与系统其余部分进行交互时，此 IP 会使用来自设计的数据来实时绘制收发器眼图。此 IP 可与设计中的用户逻辑或基于 AMD 收发器的 IP（例如，GT Wizard 或 Aurora）集成。

系统内串行 I/O 调试流程包含 3 个不同阶段：

1. In-System IBERT 核生成阶段：自定义并生成适合满足您的硬件高速串行 I/O 要求的 In-System IBERT 核。
2. 集成阶段：例化 IP，并将其集成到设计中。
3. 串行 I/O 分析阶段：与设计中包含的 In-System IBERT IP 交互，对高速串行 I/O 链路中的问题进行调试和验证。

在本章剩余部分中涵盖了有关 In-System IBERT 核生成阶段和集成阶段的详细信息。如需获取有关串行 I/O 分析阶段的详细信息，请参阅“在硬件中调试串行 I/O 设计”。

相关信息

在硬件中调试串行 I/O 设计

使用 Vivado IP 目录生成 In-System IBERT 核

设计的高速串行 I/O 接口调试的第一阶段是生成 In-System IBERT 核。

为此，请执行以下步骤：

1. 打开 Vivado IDE。
2. 在首个面板上选择“Manage IP”→“New IP Location”（管理 IP > 新建 IP 位置），在“Open IP Catalog” Wizard（打开 IP 目录向导）打开后，单击“Next”（下一步）。
3. 选择期望的部件、目标语言、目标仿真器以及 IP 位置。单击“Finish”（完成）。
4. 根据上一步中所选器件，在“Debug and Verification”→“Debug”（调试和验证 > 调试）下的“IP Catalog”（IP 目录）中可找到 1 个或多个可用的 In-System IBERT 核。
5. 双击所需的 In-System IBERT 架构，打开对应该核的“Customize IP”Wizard（自定义 IP 向导）。

按给定硬件系统要求，自定义 In-System IBERT 核。如需了解有关 In-System IBERT 核的详细信息，请参阅《In-System IBERT LogiCORE IP 产品指南》(PG246)。

在用户设计中例化 IP 并集成 In-System IBERT IP

生成 In-System IBERT IP 核后，请执行以下操作：

1. 打开顶层 RTL 文件进行编辑，添加上一步中生成的 In-System IBERT 核。
2. 复制工具生成的 In-System IBERT 核的例化模板，并在 RTL 文件中对其进行例化。
3. 将收发器端口连接至 In-System IBERT IP。

如需获取有关如何将 In-System IBERT 集成到用户设计中的详细示例，请参阅以下 IP 文档的第 5 章“设计示例”：《In-System IBERT LogiCORE IP 产品指南》(PG246)。

提示：请务必阅读《In-System IBERT LogiCORE IP 产品指南》(PG246) 的 FAQ 部分，其中列出了有关将此 IP 集成到设计中时可能遇到的问题的一些建议。

4.对设计进行综合和实现。

第 17 章

Versal 串行 I/O 硬件调试流程

AMD Versal™ 自适应 SoC 无需再生成 IBERT IP，因为使用 IBERT 所需的必要逻辑现已集成到 GTY 收发器架构内。使用 GTY 收发器的任何设计均可用于串行 I/O 硬件调试。Versal 串行 I/O 硬件调试流程具有 2 个不同阶段：

1. 设计创建。自定义并生成设计，此设计通常是使用 Versal 自适应 SoC Transceivers Wizard 或 AMD Vivado™ 中的 Versal IBERT 可配置设计示例来进行自定义和生成的，并且它会使用器件的 GTY 收发器。

   注释：在 Vivado 中可通过如下方式创建 Versal IBERT CED：单击 Vivado 的“Quick Start”（快速入门）菜单中的“Open Example Project”（打开示例工程），或者单击“File”→“Project”→“Open Example Project”（文件 > 工程 > 打开示例工程）并搜索“Versal IBERT”。

2. 串行 I/O 分析功能。使用 Vivado 硬件管理器与设计中的 GTY 收发器进行交互，以对高速串行 I/O 链路中的问题进行调试和验证。

注释：Versal IBERT 可使用来自设计的内部模式生成器（例如，PRBS）和用户数据。因此，Versal 器件不支持 In-System IBERT 核。为了获得与 In-System IBERT 相似的功能，请将模式更改为用户数据。

注释：Versal IBERT 当前不支持速率变更。此外，建议不要在位于 Vivado Serial I/O Analyzer 外部的 Transceivers Wizard 上驱动诸如 `PIN_EN` 管脚等信号，因为这可能导致不可预测的结果。之所以会产生不可预测的结果，是因为使用 IBERT 后，它会控制收发器并且可以修改收发器设置。

如需了解更多信息，请参阅《Versal Adaptive SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)。

第 18 章

在硬件中调试串行 I/O 设计

实现 IBERT 核后，您可使用运行时 Serial I/O Analyzer 功能来对硬件中的设计进行调试。仅限 IBERT 核 v3.0 版本和更高版本才能使用 Serial I/O Analyzer 功能来访问。

使用 Vivado Serial I/O Analyzer 来调试设计

AMD Vivado™ Serial I/O Analyzer 功能用于与设计中的 IBERT 调试 IP 核进行交互。要访问 Vivado Serial I/O Analyzer 功能，请单击 Flow Navigator 的“Program and Debug”（烧录和调试）部分中的“Open Hardware Manager”（打开硬件管理器）按钮。

对硬件中的设计进行调试的步骤如下：

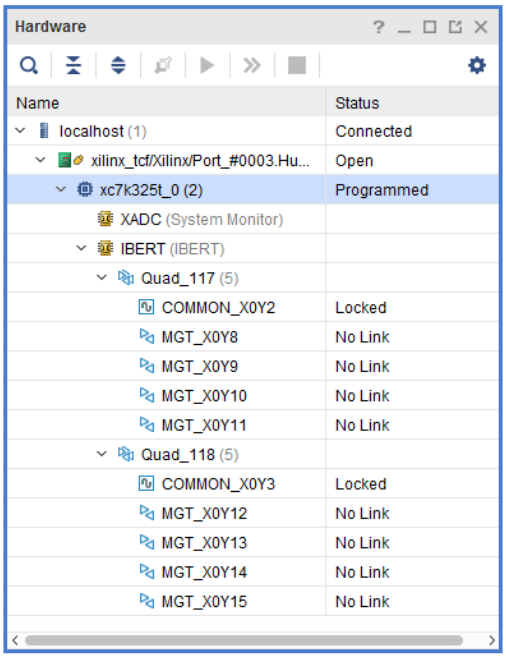
1. 连接到硬件目标并使用比特文件对 FPGA 进行烧录。
2. 创建链路。
3. 修改链路设置并检验状态。
4. 按需运行扫描。

连接至硬件目标并执行器件烧录

在调试前对 FPGA 或自适应 SoC 进行烧录的步骤与“对 FPGA 或自适应 SoC 进行烧录”中所述的步骤完全相同。使用包含 IBERT 核的 `.pdi` 文件完成器件烧录后，“Hardware”（硬件）窗口就会显示 IBERT 核的组件，并在其右侧括号内显示扫描器件时检测到的 RTL 实例名称（如下图所示）。

重要提示！如果在您的设计中为 UltraScale 和 UltraScale+ 设计使用了 In-System IBERT IP，那么您将在“Hardware”窗口中看到所检测到的 In-System IBERT 核。

图 175：**显示 IBERT 核的“Hardware”窗口**



相关信息

器件烧录

创建链路和链路组

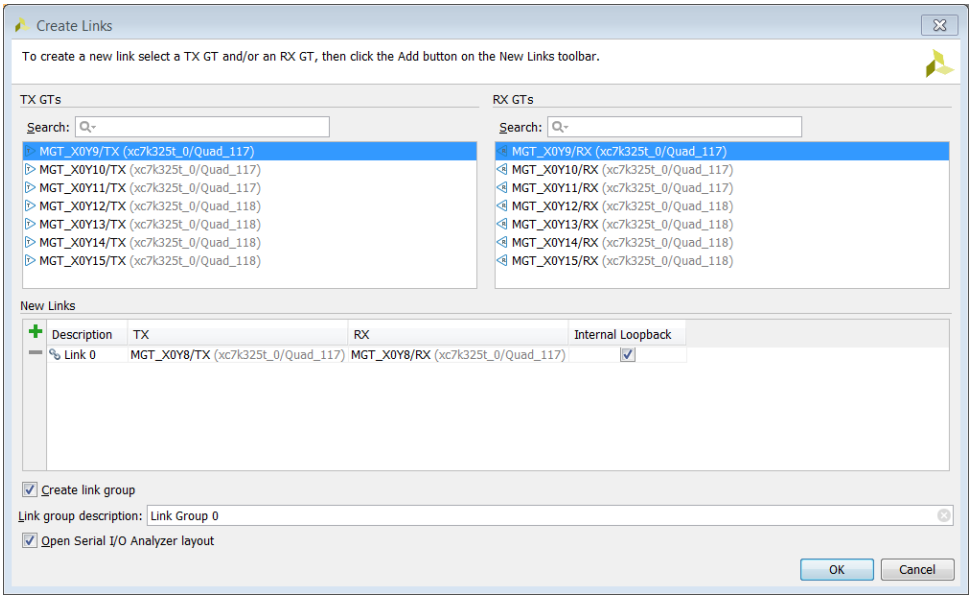
设计中存在的 IBERT 核会显示在“Hardware”（硬件）窗口中的目标器件下。如果未显示此核，请右键单击器件并选中“Refresh Hardware”（刷新硬件）命令。这样将重新扫描 FPGA 并刷新“Hardware”窗口。

注释：如果烧录和/或刷新 FPGA 器件后仍未显示 IBERT 核，请检查并确保已使用正确的 BIT 文件完成了器件烧录。此外，请检查并确保已实现的设计包含 IBERT v3.0 核。

Vivado Serial I/O Analyzer 功能是围绕链路概念构建的。链路是开发板上通道的模拟链路，具有发射器和接收器。发射器和接收器可以采用相同或不同 GT、位于相同或不同器件上或者采用相同或不同架构。由于链路必须同时与发射器和接收器关联，因此不支持将外部模式生成器连接到单一 GT 接收器。要创建 1 条或多条链路，请转至 Vivado 中的“Links”（链路）选项卡，并单击“Create Links”（创建链路）按钮或者右键单击并选中“Create Links”。这将显示“Create Links”对话框，如下图所示。

检测到 IBERT 核时，硬件管理器会发现不存在任何链路，并在顶部显示绿色条幅。请单击“Create Links”以打开对话框，如下图所示。

图 176：**“Create Links”对话框**



从可用列表中选择 TX 和/或 RX。或者在搜索字段中输入字符串以缩小列表范围。然后单击“+”（添加）按钮，以将链路添加到列表中。针对所有目标链路重复此过程。

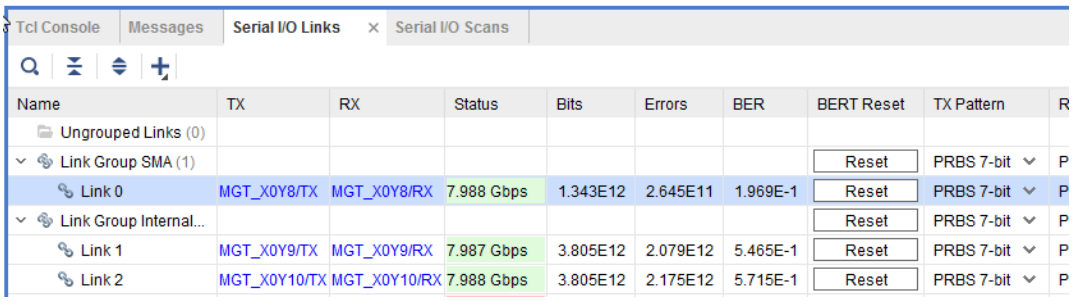
重要提示！任一给定 TX 或 RX 端点只能属于单一链路。

链路还可包含在链路组中。默认情况下，所有新链路都分组在一起。您可通过取消选中“Create link group”（创建链路组）来避免将链路添加到同一组中。链路组名称可在“Link group description”（链路组描述）字段中指定。

使用“Links”窗口查看和更改链路设置

创建链路后，就会将其添加到“Links”（链路）视图（请参阅下图）中，该视图是更改链路设置和查看状态的主要方法，也是最佳方法。

图 177：**“Links”窗口**



“Links”窗口中的每一行都对应 1 条链路。默认情况下，常用的实用状态和控制均处于启用状态，因此，链路运行状况一目了然。下表显示了“Links”窗口的表格列中可查看的各项设置。

表 28：**“Links”窗口设置**

链路视图列名	描述
“Name”（名称）	链路名称
TX	发射器的 GT 位置
RX	接收器的 GT 位置
“Status”（状态）	是否已链接（表示 RX 数据是否按期望方式传入）。“Status”可显示测量所得线速率。如果未链接，则显示“No Link”（无链路）。
“Bits”（位）	测量所得接收到的位数。
“Errors”（错误）	接收器测量所得的位错误数量。
BER	误码率 (BER) = (1 + 错误数) / (位数)。
“BERT Reset”（BERT 复位）	将收到的比特数和误码计数器复位。
“RX Pattern”（RX 模式）	选择接收器期望的模式。
“TX Pattern”（TX 模式）	选择发射器发送的模式。
“TX Pre-Cursor”（TX 前标）	选择发射器上的前标加重。
“TX Post-Cursor”（TX 后标）	选择发射器上的后标加重。
“TX Diff Swing”（TX 差分摆幅）	选择发射器的差分摆幅值。
“DFE Enabled”（DFE 已启用）	选择是否在接收器上启用“Decision Feedback Equalizer”（判定反馈均衡器），它并非对所有架构都可用。
“Inject Error”（插入误码）	在发射路径中插入单位错误。
“TX Reset”（TX 复位）	将发射器复位。
“RX Reset”（RX 复位）	将接收器和 BERT 计数器复位（请参阅“BERT 复位”）。
“Loopback Mode”（环回模式）	选择接收器 GT 上的环回模式。 **警告！**根据系统拓扑结构，更改该值可能会影响链路状态。
“Termination Voltage”（终端电压）	选择接收器的终端电压。
“RX Common Mode”（RX 共模）	选择接收器的 RX 共模设置。
“TXUSERCLK Freq”（TXUSERCLK 频率）	显示测量所得 TXUSERCLK 频率（以 MHz 为单位）。
“TXUSERCLK2 Freq”（TXUSERCLK2 频率）	显示测量所得 TXUSERCLK2 频率（以 MHz 为单位）。
“RXUSERCLK Freq”（RXUSERCLK 频率）	显示测量所得 RXUSERCLK 频率（以 MHz 为单位）。
“RXUSERCLK2 Freq”（RXUSERCLK2 频率）	显示测量所得 RXUSERCLK2 频率（以 MHz 为单位）。
“TX Polarity Invert”（RX 极性反相）	将发射的数据的极性反相。
“RX Polarity Invert”（RX 极性反相）	将接收的数据的极性反相。

可通过更改链路组行中的设置来更改链路组中所有链路的任一给定属性的值。例如，将“Link Group 0”行中的“TX Pattern”更改为“PRBS 7-bit”即可将所有链路的 TX 模式更改为“PRBS 7-bit”。如果组中并非所有链路都采用相同设置，那么在链路组行中，该列会显示“Multiple”。

当设计中使用 In-System IBERT IP 时，仅适用一小部分链路设置。下表列出了适用的链路设置。

表 29：**In-System IBERT 的“Link”（链路）窗口设置**

链路视图列名	描述
TX	发射器的 GT 位置
RX	接收器的 GT 位置
“TX Pre-Curser”（TX 前标）	选择发射器上的前标加重。
“TX Post-Cursor”（TX 后标）	选择发射器上的后标加重。
“TX Diff Swing”（TX 差分摆幅）	选择发射器的差分摆幅值。
“DFE Enabled”（DFE 已启用）	选择是否在接收器上启用“Decision Feedback Equalizer”（判定反馈均衡器），它并非对所有架构都可用。

创建和运行链路扫描

要分析给定链路的裕度，通常最好使用 AMD 7 系列 FPGA 收发器的专用 Eye Scan（眼图扫描）硬件来运行链路扫描。Vivado Serial I/O Analyzer 功能支持您定义、运行、保存和重新调用链路扫描。

扫描在链路上运行。要创建扫描，请在“Link”（链路）窗口上选中链路，然后右键单击并选择“Create Sweep”（创建清扫），或者单击“Link”窗口工具栏中的“Create Sweep”按钮。这样即可打开“Create Scan”（创建扫描）对话框（请参阅下图）。“Create Scan”对话框可显示用于执行扫描的设置，如下表所示。

可生成 2 种类型的扫描：“2D Eyescan”（2D 眼图扫描）或“1D Bathtub Plot”（1D 浴缸图）。这两种扫描均使用“Create Scan”（创建扫描）对话框中指定的设置，如下所示。以下对话框中的“Scan type”（扫描类型）字段可用于判定所生成的扫描类型。

图 178：**“Create Scan”对话框**

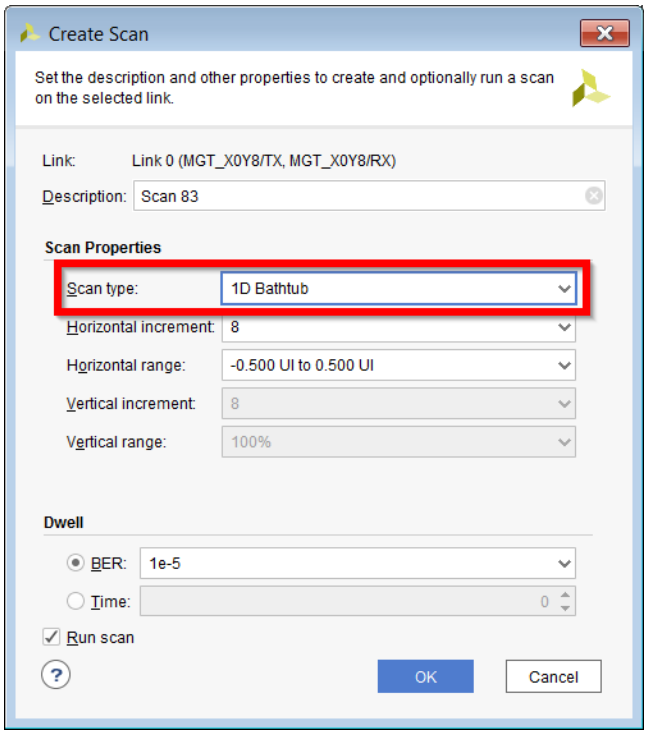


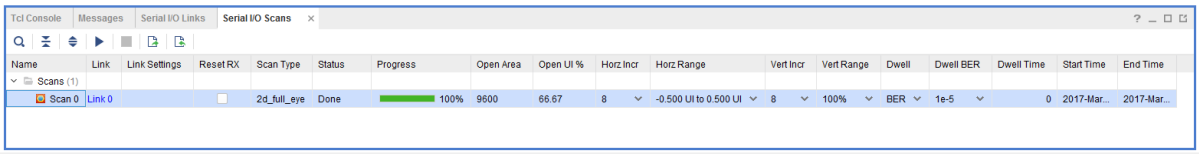
表 30：**扫描设置**

扫描设置	描述
描述	用户定义的扫描名称。
“Scan Type”（扫描类型）	要运行的扫描类型。类型包括：“2D Eyescan Plot”（2D 眼图）或“1D Bathtub Plot”（1D 浴缸图）。
“Horizontal Increment”（水平增量）	允许您选择以较低的分辨率扫描眼图，但通过跳过水平代码来提速。
“Horizontal Range”（水平范围）	减小水平范围可提升扫描速度。默认情况下，将扫描整个眼图（参考眼图中心，按单位间隔的 -1/2 到 +1/2 范围）。
“Vertical Increment”（垂直增量）	允许您选择以较低的分辨率扫描眼图，但通过跳过垂直代码来提速。
“Vertical Range”（垂直范围）	减小垂直范围可提升扫描速度。默认情况下，将扫描整个眼图。
“Dwell BER”（停顿误码率）	对图表中每个点进行一段时间的扫描。“Dwell BER”允许您通过选择期望的误码率来选择扫描深度。
“Dwell Time”（停顿时间）	“Dwell Time”允许您通过输入期望的时间（秒）来选择扫描深度。 Dwell Time 设置在使用 In-System IBERT IP 的设计上不予支持。

默认情况下，创建扫描后立即运行扫描。如果不想运行扫描，且只需定义扫描即可，那么请取消勾选“Run Scan”（运行扫描）复选框。

如果创建但不运行扫描，则可稍后再运行，或者也可以通过在“Scans”窗口中右键单击任一扫描并选中“Run Scan”来运行该扫描（请参阅下图）。运行扫描时，可通过右键单击扫描并单击“Stop Scan”（停止扫描），或者通过单击“Scans”（扫描）窗口工具栏中的“Stop Scan”（停止扫描）按钮，以将其提前停止运行。

图 179：**“Scans”窗口**



创建和运行链路清扫

要分析给定链路的裕度，利用不同 MGT 设置来多次运行链路扫描是很有效的。这样有助于判定最佳设置。Vivado Serial I/O Analyzer 功能支持您定义、运行、保存和重新调用链路清扫，链路清扫是由多次链路扫描集合而成的。

每条链路对应一次清扫。要创建清扫，请选中“Link”（链路）窗口中的链路，然后右键单击并选择“Create Sweep”（创建清扫），或者单击“Link”窗口工具栏中的“Create Sweep”按钮。这样会显示“Create Sweep”（创建清扫）对话框，此对话框与“Create Scan”（创建扫描）对话框相似，差别在于前者具有额外的选项用于定义要清扫的属性以及清扫方式。

图 180：**“Create Sweep”对话框**

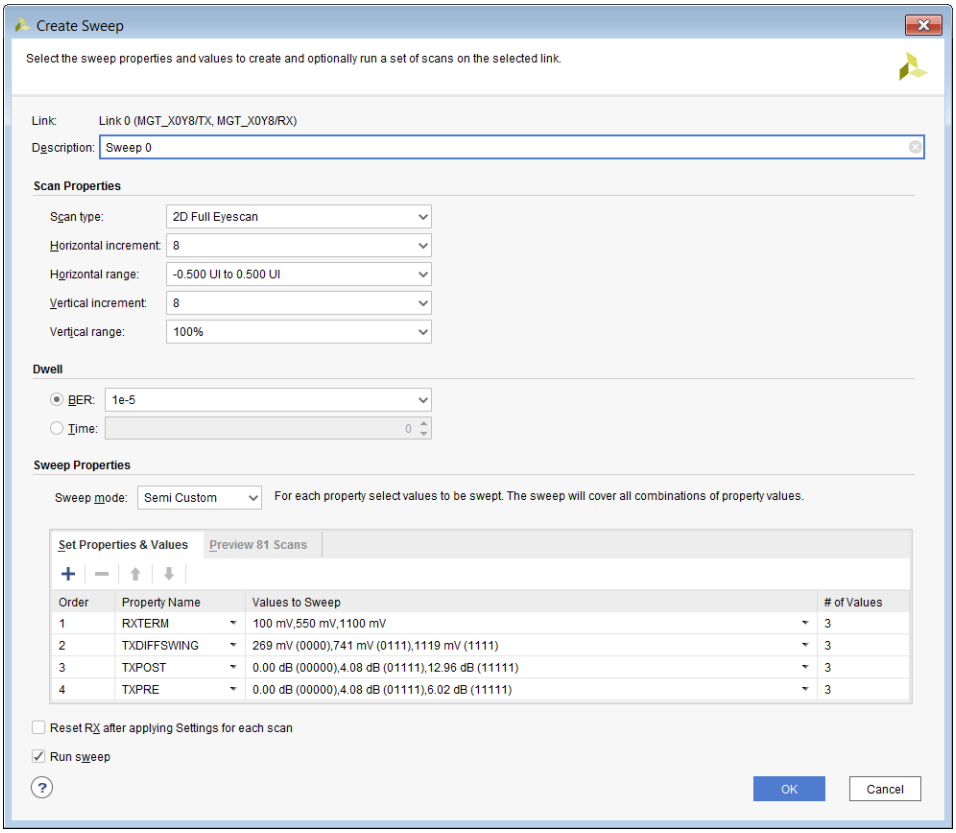


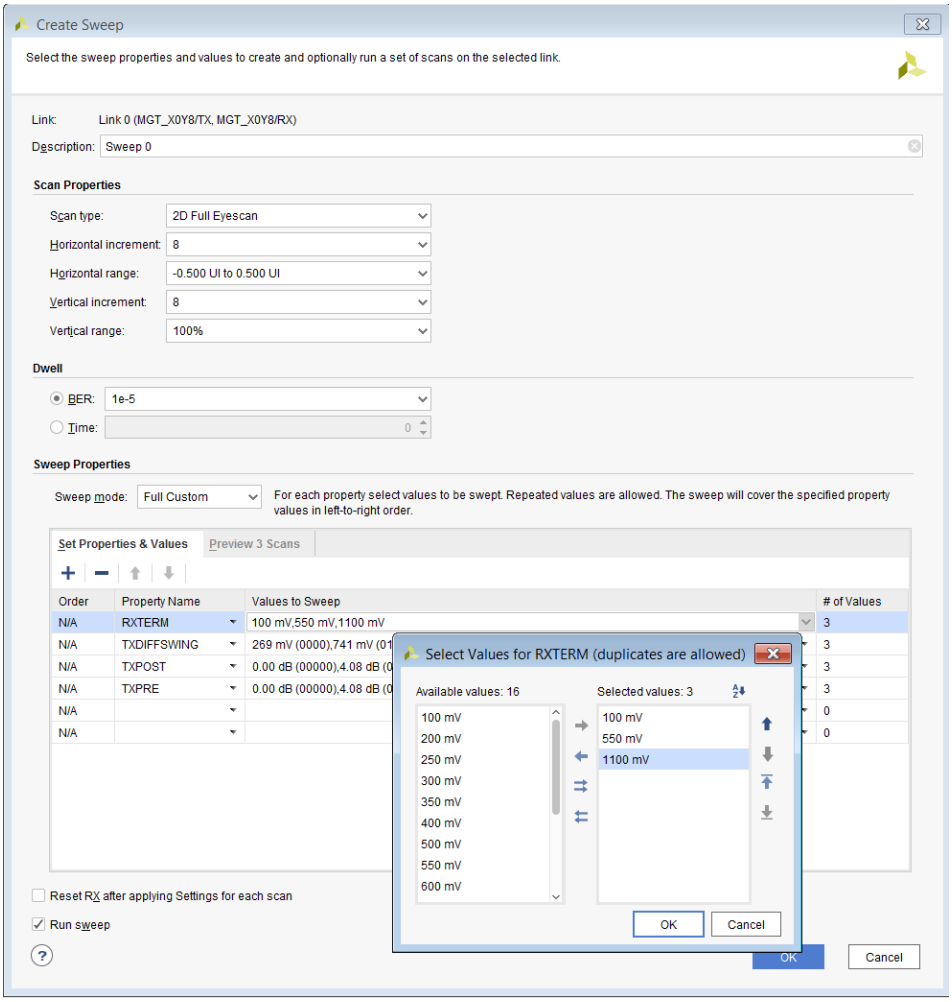
表 31：**清扫设置**

清扫设置	描述
描述	用户定义的清扫名称。
“Scan Type”（扫描类型）	要运行的扫描类型。类型包括：“2D Eyescan Plot”（2D 眼图）或“1D Bathtub Plot”（1D 浴缸图）。
“Horizontal Increment”（水平增量）	允许您以较低的分辨率扫描眼图，但通过跳过水平代码来提速。
“Horizontal Range”（水平范围）	减小水平范围可提升扫描速度。默认情况下，将扫描整个眼图（参考眼图中心，按单位间隔的 -1/2 到 +1/2 范围）。
“Vertical Increment”（垂直增量）	允许用户选择以较低的分辨率扫描眼图，但通过跳过垂直代码来提速。
“Vertical Range”（垂直范围）	减小垂直范围可提升扫描速度。默认情况下，将扫描整个眼图。
“Dwell BER”（停顿误码率）	对图表中每个点进行一段时间的扫描。“Dwell BER”允许您通过选择期望的误码率 (BER) 来选择扫描深度。
“Dwell Time”（停顿时间）	“Dwell Time”允许您通过输入期望的时间（秒）来选择扫描深度。
“Sweep Mode”（清扫模式）	要运行的清扫类型。选项包括：“Semi Custom”（半定制）、“Full Custom”（全定制）和“Exhaustive”（详尽）。

选定这些设置后，下一步是选择“Sweep Properties”（清扫属性）。所有可写链路属性都可接受清扫。要添加属性，请单击左侧“+”按钮以在表格中添加另一行。单击“Property Name”（属性名称）即可选择要清扫的属性。

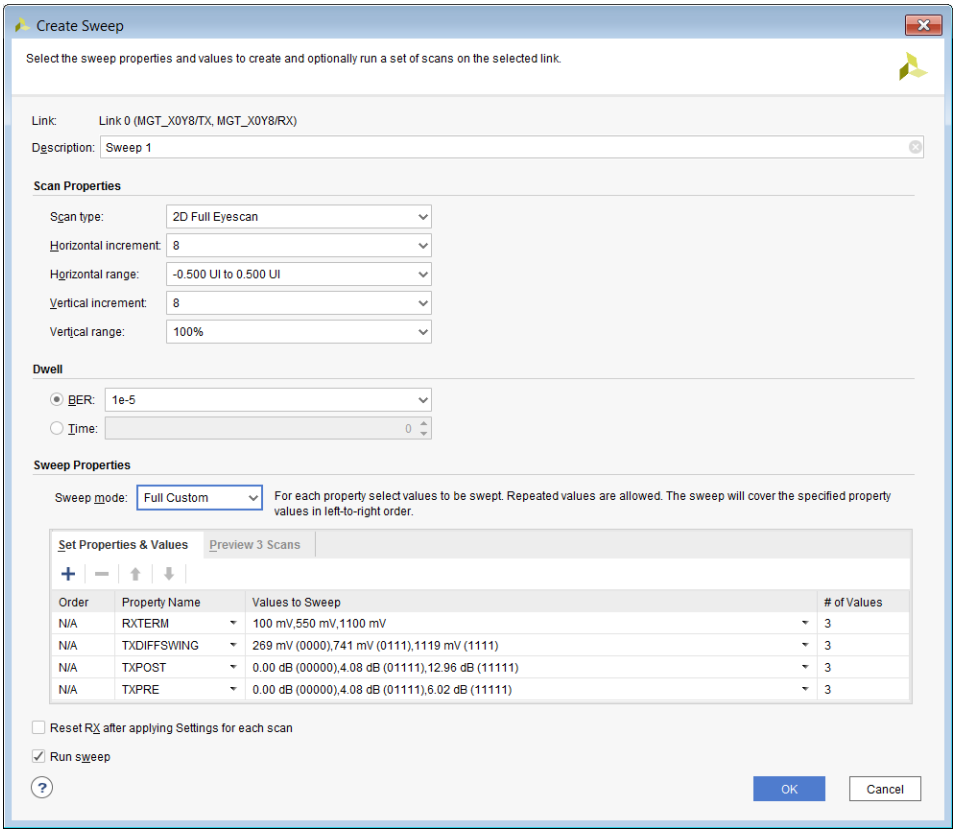
要更改值，请单击“Values to Sweep Cell”（单元清扫值），并使用选择器来选择要清扫的值。如果属性不含枚举值，请在提供的文本区域的每一行上输入 1 个十六进制值。

图 181：**Values to Sweep Cell**



- 在下图所示的“Semi Custom”模式下，将针对每一次清扫定义每一种属性组合，并且将根据清扫属性来执行清扫。可通过选中“Preview & Scans”（预览并扫描）选项卡来预览执行的清扫数量以及清扫顺序。
- 在“Full Custom”（全定制）模式下，列出的每个属性的第一个选项用于首次扫描，每个属性的第二个选项用于第二次扫描，以此类推。如果任一属性所含选项数少于其他属性，则最后一个选项将用于所有后续扫描。如果属性选项全部相同，但采用“Full Custom”作为清扫模式，那么只能执行 3 次清扫。

图 182：**“Sweep Properties”对话框**



- 在“Exhaustive”模式下，“Values to Sweep”不可编辑，因为针对任一给定属性将选中所有值。

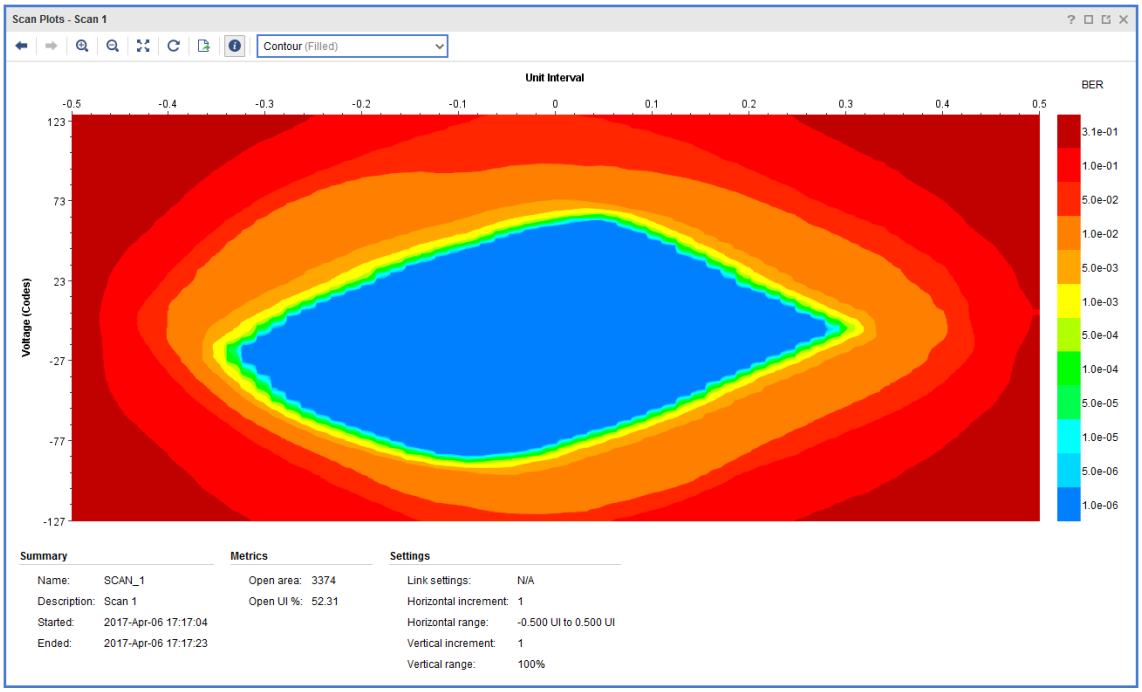
所有属性都完成设置后，要按顺序运行每一次扫描，请保持“Run Sweep”（运行清扫）处于选中状态。单击“OK”后就会在“Scan”（扫描）窗口中详细罗列扫描列表。

清扫期间，在“Scan”窗口中将跟踪进度，并显示最新的清扫结果。

显示和浏览扫描图

创建扫描后，它会为扫描自动启动“Scan Plots”（扫描图）窗口。对于 2D 眼图扫描，扫描图为 BER 值组成的热图。

图 183：**“Scan Plot”窗口**



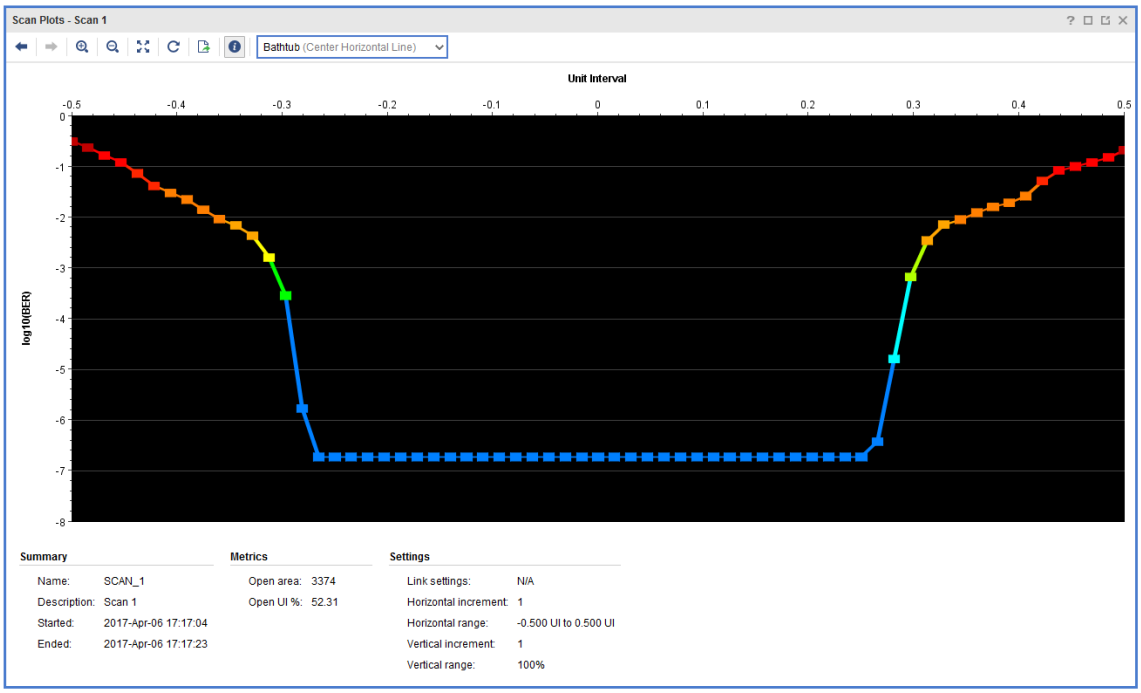
就像 Vivado IDE 中显示的其他图表一样，眼图扫描图窗口中的缩放鼠标手势如下：

- “Zoom Area”（缩放区域）：左键单击并从左上向右下拖动
- “Zoom Fit”（缩放适应）：左键单击并从右下向左上拖动
- “Zoom In”（放大）：左键单击并从右上向左下拖动
- “Zoom Out”（缩小）：左键单击并从左下向右上拖动

并且当鼠标置于图上时，当前水平和垂直代码以及扫描所得 BER 值都会显示在工具提示中。您也可以单击扫描图窗口中的“Plot Type”（图类型）按钮并选择“Show Contour (filled), Show Contour (lines), Bathtub (Center Horizontal Line), and Heat Map”（显示等高线（已填充）、显示等高线（线条）、浴缸图（中间水平线）和热图）来更改扫描图的类型。

在扫描图底部会显示摘要视图，其中显示了扫描设置以及扫描执行时间等基本信息。在执行 2D 眼图扫描期间，将计算扫描中不含任何错误的像素数量（将水平和垂直增量一并纳入考量），此结果将显示为“Open Area”（开放区域）。“Scan”（扫描）窗口内容默认情况下按“Open Area”排序，因此开放区域最大的扫描显示在顶部。下图为上图所示扫描的浴缸图。

图 184：**浴缸图**



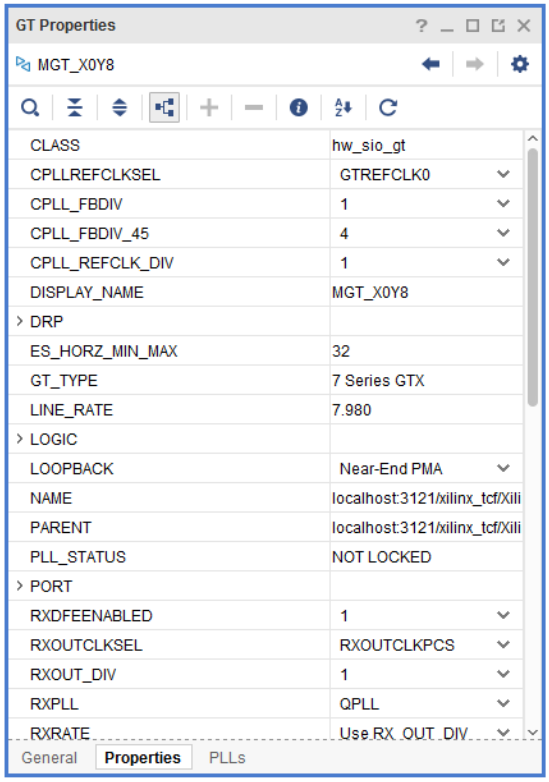
将扫描结果写入文件

如果由于执行了部分或完整 2D 眼图扫描导致存在扫描数据，那么可通过单击“Scan”（扫描）窗口中的“Write Scan”（写入扫描）按钮，将这些结果写入 CSV 文件。这样即可将扫描结果保存到逗号分隔格式的文件，并将 BER 值归入同一个区块（经复制扫描图所得）内。

“Properties”窗口

只要在“Hardware”（硬件）窗口中选中 GT 或 COMMON 块、在“Link”（链接）窗口中选中链接，或者在“Scan”（扫描）窗口中选中扫描，那么就会在“Properties”（属性）窗口中显示该对象的属性。对于 GT 和 COMMON，该窗口可显示这些对象的所有属性、端口和其他设置。这些设置可在“Properties”（属性）窗口中更改（请参阅下图），或者也可以使用 Tcl 命令来更改和落实属性。部分属性为只读，无法更改。

图 185：**“Properties”窗口**



Serial I/O Analyzer Tcl 对象和命令的描述

您可使用 Tcl 命令与所测试的硬件进行交互。硬件可组织为一组分层式第一类 Tcl 对象（请参阅下表）。

表 32：**Serial I/O Analyzer Tcl 对象**

Tcl 对象	描述
hw_sio_ibert	引用 IBERT 核的对象。每个 IBERT 对象都可包含 1 个或多个与之关联的 hw_sio_gt 或 hw_sio_common 对象。
hw_sio_gt	引用单个 AMD 千兆位收发器 (GT) 的对象。
hw_sio_gtgroups	引用 GT 逻辑分组的对象，可采用“Quad”（四通道）或“Octal”（八通道）。
hw_sio_common	引用 COMMON 块的对象。
hw_sio_tx	引用 hw_sio_gt 的发射器侧的对象。仅限 TX 相关端口、属性和逻辑属性才会流入 hw_sio_tx。
hw_sio_rx	引用 hw_sio_gt 的接收器侧的对象。仅限 RX 相关端口、属性和逻辑属性才会流入 hw_sio_rx。
hw_sio_pll	引用 hw_sio_gt 或 hw_sio_common 对象中的 PLL 的对象。仅限相关端口、属性和逻辑属性才会流入 hw_sio_pll。
hw_sio_link	引用链路（TX-RX 对）的对象。 链路也可以仅含 TX 或仅含 RX。
hw_sio_linkgroup	引用一组链路的对象。
hw_sio_scan	引用裕度分析扫描的对象。

如需了解有关硬件管理器命令的更多信息，请在 Tcl 控制台中运行 `help -category hardware` Tcl 命令。

用于访问硬件的 Tcl 命令的描述

下表包含用于与 IBERT 核进行交互的所有 Tcl 命令的描述。

重要提示！使用 get_property 命令或 set_property 命令并不能从 IBERT 核读取信息，也无法向该核写入信息。您必须使用 refresh_hw_sio 命令和 commit_hw_sio 命令来分别从硬件读取信息和向硬件写入信息。

表 33：**hw_server Tcl 命令的描述**

Tcl 命令	描述
refresh_hw_sio	从提供的对象中读出属性值。适用于引用该硬件的任意 hw_sio 对象。
commit_hw_sio	将属性更改写入硬件。适用于引用该硬件的任意 hw_sio 对象。

hw_sio_link Tcl 命令的描述

下表包含用于与链路进行交互的所有 Tcl 命令的描述。

表 34：**hw_sio_link Tcl 命令的描述**

Tcl 命令	描述
create_hw_sio_link	创建 hw_sio_link 对象，其中包含给定 hw_sio_rx 和/或 hw_sio_tx 对象。
remove_hw_sio_link	删除给定链路。
get_hw_sio_links	获取给定对象的 hw_sio_links 列表。

hw_sio_linkgroup Tcl 命令的描述

下表包含与链路组 (linkgroup) 进行交互的所有 Tcl 命令的描述。

表 35：**hw_sio_linkgroup Tcl 命令的描述**

Tcl 命令	描述
create_hw_sio_linkgroup	创建 hw_sio_linkgroup 对象，其中包含多个 hw_sio_link 对象。
remove_hw_sio_linkgroup	删除给定链路组。
get_hw_sio_linkgroups	获取给定对象的 hw_sio_linkgroups 列表。

hw_sio_scan Tcl 命令描述

下表包含与扫描进行交互的所有 Tcl 命令的描述。

表 36：**hw_sio_scan Tcl 命令的描述**

Tcl 命令	描述
create_hw_sio_scan	创建扫描对象。
remove_hw_sio_scan	删除扫描对象。
run_hw_sio_scan	运行扫描。
stop_hw_sio_scan	停止扫描。
wait_on_hw_sio_scan	阻止 Tcl 控制台提示，直至给定 run_hw_sio_scan 操作完成为止。

表 36：**hw_sio_scan Tcl 命令的描述** (续)

Tcl 命令	描述
display_hw_sio_scan	在扫描图中显示部分扫描或完整扫描。
write_hw_sio_scan	将扫描数据写入文件。
read_hw_sio_scan	将扫描数据从文件读取到扫描对象中。
get_hw_sio_scans	获取 hw_sio_scan 对象列表。

用于获取对象的 Tcl 命令的描述

下表包含用于获取串行 I/O 对象的所有 Tcl 命令的描述。

表 37：**用于获取对象的 Tcl 命令的描述**

Tcl 命令	描述
get_hw_sio_iberts	获取 IBERT 对象列表。
get_hw_sio_gts	获取 GT 列表。
get_hw_sio_commons	获取 COMMON 块列表。
get_hw_sio_txs	获取发射器列表。
get_hw_sio_rxs	获取接收器列表。
get_hw_sio_plls	获取 PLL 列表。
get_hw_sio_links	获取链路列表。
get_hw_sio_linkgroups	获取链路组列表。
get_hw_sio_scans	获取扫描列表。

使用 Tcl 命令来执行 IBERT 测量

以下示例提供了与下列系统示例进行交互的 Tcl 命令脚本

- 1 条 KC705 评估板的 Digilent JTAG-SMT1 线（序列号 12345），可通过 localhost:3121 上运行的 hw_server 来访问
- 在 KC705 评估板上的 XC7K325T 器件中运行的设计内包含单个 IBERT 核
- IBERT 核已启用 Quad 117 和 Quad 118

Tcl 命令脚本示例

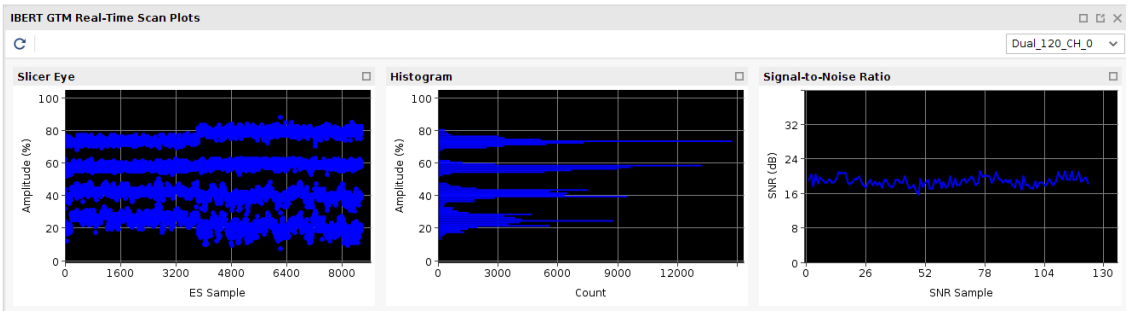
```
# Connect to the Digilent Cable on localhost:3121 connect_hw_server -url
localhost:3121 current_hw_target [get_hw_targets */digilent_plugin/
SN:12345] open_hw_target # Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0] refresh_hw_device -
update_hw_probes false [lindex [get_hw_devices] 0] set_property
PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0] program_hw_devices
[lindex [get_hw_devices] 0] refresh_hw_device [lindex [get_hw_devices] 0] #
Set Up Link on first GT set tx0 [lindex [get_hw_sio_txs] 0] set rx0 [lindex
[get_hw_sio_rxs] 0] set link0 [create_hw_sio_link $tx0 $rx0] set_property
DESCRIPTION {Link 0} [get_hw_sio_links $link0] # Set link to use PCS
```

```
Loopback, and write to hardware set_property LOOPBACK "Near-End PCS" $link0
commit_hw_sio $link0 # Create, run, display and save scan set scan0
[create_hw_sio_scan 2d_full_eye [get_hw_sio_rxs -of $link0]]
run_hw_sio_scan $scan0 display_hw_sio_scan $scan0 write_hw_sio_scan
"scan0.csv" $scan0
```

查看 Slicer 眼图、直方图和信噪比图（仅限 GTM 收发器）

由于 GTM 接收器基于 ADC，导致无法使用先前系列的收发器（例如，GTH 或 GTY 收发器）中所使用的传统眼图。因此，GTM 的 IBERT 仪表板可显示 3 种图：Slicer 眼图、直方图和信噪比 (SNR) 图，而无法显示传统扫描图窗口。

图 186：**Slicer 眼图、直方图和信噪比图**



创建链接后，将显示此链接的 Slicer 眼图、直方图和信噪比 (SNR) 图。如果创建了多个链接，那么可通过选中右上角的“DUAL”和“Channel”（通道）来更改活动链接。

如需了解有关 Slicer 眼图和 GTM 收发器架构的更多信息，请参阅《UltraScale + FPGA GTM 收发器用户指南》(UG581)。

如需了解有关 IBERT GTM 的更多信息，请参阅《IBERT for UltraScale GTM Transceivers LogiCORE IP 产品指南》(PG342)。

附录 A

器件配置比特流或 PDI 设置

7 系列比特流设置

下表所示 7 系列 器件的器件配置设置可搭配 `set_property <Setting> <Value> [current_design]` AMD Vivado™ 工具 Tcl 命令一起使用。

注释： BPI 的比特流设置对于 Spartan™ 7 器件无效。

表 38：**7 系列比特流设置**

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3 或 4	此设置用于将 BPI 配置与闪存器件中的页面模式操作的时序进行同步。它允许您为首个页面的有效读取设置周期数。BPI_page_size 必须设置为 4 或 8，这样该选项才可用。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	对于 BPI 配置，该选项允许您指定页面大小，此大小对应于闪存每个页面所需读取数。
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1 和 Type2	此设置用于为不同类型的 BPI 闪存器件设置 BPI 同步配置模式。 Disable（默认值）即禁用同步配置模式。 Type1 启用同步配置模式和设置以支持 Micron G18(F) 系列。 Type2 启用同步配置模式和设置以支持 Micron (Numonyx) P30 和 P33 系列。
BITSTREAM.CONFIG.CCLKPIN	Pullup	Pullup 和 Pullnone	此设置用于向 Cclk 管脚添加内部上拉。Pullnone 设置则禁用上拉。
BITSTREAM.CONFIG.CONFIGFALLBACK	Disable	Disable 和 Enable	当配置尝试失败时，启用或禁用默认比特流的加载。 如果所使用的多重启动 (MultiBoot) 解决方案设置为 BITSTREAM.CONFIG.NEXT_CONFIG_ADDR，则启用 BITSTREAM.CONFIG.FALLBACK 设置。 在 Virtex 7 HT 器件中不支持回退多重启动。
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、16、22、26、33、40、50 和 66	在主模式下配置时，使用内部振荡器来生成配置时钟 Cclk。该选项用于选择 Cclk 的速率。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Continuous 和 Quiet	此设置用于控制数控阻抗电路尝试对 DCI IOSTANDARD 执行阻抗匹配更新的频率。
BITSTREAM.CONFIG.DONEPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于向 DONE 管脚添加内部上拉。Pullnone 设置则禁用上拉。仅当您要将外部上拉电阻器连接到此管脚时，才使用 DonePin。如果您不使用 DonePin，则自动连接内部上拉电阻器。

表 38：**7 系列比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable、Div-1、Div-2、Div-4 和 Div-8	此设置允许使用外部时钟作为所有主模式的配置时钟。此外部时钟必须连接到两用 EMCCLK 管脚。
BITSTREAM.CONFIG.INITPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于指定是要将 Pullup 电阻器添加到 INIT 管脚，还是保留 INIT 管脚处于浮动状态。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable 和 Disable	设为“Enabled”（启用）后，如果检测到配置错误，则 INIT_B 管脚断言为“0”。
BITSTREAM.CONFIG.M0PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M0 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M0 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M1PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M1 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M1 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M2PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M2 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M2 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	无	<字符串>	此设置用于在 MultiBoot 设置（存储在 WBSTAR 寄存器中）中设置下一次配置的起始地址。
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable 和 Disable	设置为 Disable 时，将从 `.bit` 文件中移除 IPROG 命令。这样即可允许在上电时加载黄金镜像，而不是跳转到多重启动配置中的多重启动镜像。
BITSTREAM.CONFIG.PERSIST	No	No 和 Yes	此设置用于在完成配置后，保留对多功能配置管脚的配置逻辑访问权。此设置主要用于在配置后保留 SelectMAP 端口用于回读访问，但也可配合任意配置模式使用。对于 JTAG 配置则无需保留 (PERSIST)，因为 JTAG 端口是专用端口且始终可用。PERSIST 和 ICAP 不能同时使用。 请参阅用户指南以获取相关说明。针对使用 SelectMAP 配置管脚的“Readback”（回读）和“Partial Reconfiguration”（部分重配置）操作，需使用 PERSIST，并且使用 SelectMAP 模式或串行模式时，也应使用 PERSIST。
BITSTREAM.CONFIG.REVISIONSELECT	00	00、01、10 或 11	此设置用于在温态启动起始地址 (WBSTAR) 寄存器中为下一次温态启动指定 RS[1:0] 设置的内部值。
BITSTREAM.CONFIG.REVISIONSELECT_ TRISTATE	Disable	Disable 和 Enable	此设置用于通过在温态启动起始地址 (WBSTAR) 中设置相应选项来指定是否启用 RS[1:0] 三态。 0：启用 RS 三态 1：禁用 RS 三态
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable 和 Disable	此设置用于启用或禁用 SelectMAP 模式“Abort”（异常中止）序列。如果禁用，则忽略器件管脚上的 Abort 序列。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No 和 Yes	此设置启用 SPI 32 位地址样式，对于存储空间不小于 256 Mb 的 SPI 器件，此设置是必需的。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2 和 4	针对来自第三方 SPI 闪存器件的“Master SPI”（主 SPI）配置，将 SPI 总线设置为“Dual (x2)”（双通道）或“Quad (x4)”（四通道）模式。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No 和 Yes	此设置用于将 FPGA 设置为使用下降沿时钟进行 SPI 数据捕获。这样可改进时序裕度，并能够提升配置的时钟速率。
BITSTREAM.CONFIG.TCKPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TCK 管脚添加上拉、下拉或两者都不添加，此管脚是 JTAG 测试时钟。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDIPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDI 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和 JTAG 寄存器的串行数据输入。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。

表 38：**7 系列比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.TDOPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDO 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和数据寄存器的串行数据输出。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TIMER_CFG	无	<8 个数字的十六进制字符串>	在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_USR 同时使用。
BITSTREAM.CONFIG.TIMER_USR	0x00000000	<8 个数字的十六进制字符串>	在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_CFG 同时使用。
BITSTREAM.CONFIG.TMSPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TMS 管脚添加上拉、下拉或两者都不添加，此管脚是 TAP 控制器的模式输入信号。TAP 控制器可为 JTAG 提供控制逻辑。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown、Pullup 和 Pullnone	此设置用于向未使用的 SelectIO™ 管脚 (IOB) 添加上拉、下拉或者两者都不添加。它对于专用配置管脚无效。专用配置管脚列表因架构而异。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	<8 个数字的十六进制字符串>	此设置用于识别实现的版本。您可在“User ID”（用户 ID）寄存器中添加最多 1 个含 8 个数字的十六进制字符串。
BITSTREAM.CONFIG.USR_ACCESS	无	None、<8 个数字的十六进制字符串> 和 TIMESTAMP	此设置用于将 1 个含 8 个数字的十六进制字符串或时间戳写入 AXSS 配置寄存器。时间戳值的格式为 ddddd MMMM yyyyyy hhhhh mmmmmm ssssss：对应日、月、年（2000 年 = 00000）、小时、分钟、秒。FPGA 互连结构可通过 USR_ACCESS 原语直接访问此寄存器的内容。
BITSTREAM.ENCRYPTION.ENCRYPT	No	No 和 Yes	加密比特流。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram 和 efuse	此设置用于判断将使用的 AES 加密密钥的位置：密钥来自于电池供电式 RAM (BBRAM) 或 eFUSE 寄存器。 仅当“Encrypt”选项设置为“True”时，该属性才可用。
BITSTREAM.ENCRYPTION.HKEY	无	<十六进制字符串>	HKey 用于为比特流加密设置 HMAC 身份验证密钥。7 系列器件具有片上比特流密钥式散列消息验证码 (HMAC) 算法，此算法在硬件中实现，以在 AES 解密基础上提供额外的安全性。这些器件需要 AES 和 HMAC 密钥才能对比特流执行加载、修改、拦截或克隆。 要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.KEY0	无	<十六进制字符串>	Key0 用于为比特流加密设置 AES 加密密钥。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.KEYFILE	无	<字符串>	此设置用于指定输入加密文件的名称（含 `.nky` 文件扩展名）。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.STARTCBC	无	<32 位十六进制字符串>	此设置用于设置起始密码分组链接 (CBC) 值。
BITSTREAM.GENERAL.COMPRESS	False	True 和 False	比特流中的多帧写入功能用于减小比特流的大小，而不是用于减小比特流 (`.bit`) 文件的大小。使用 COMPRESS 并不保证比特流大小会缩小。
BITSTREAM.GENERAL.CRC	Enable	Enable 和 Disable	此设置用于控制比特流中循环冗余校验 (CRC) 值的生成。设为启用后，可根据比特流内容计算出唯一的 CRC 值。如果计算所得的 CRC 值与比特流中的 CRC 值不匹配，那么器件将无法进行配置。禁用 CRC 时，将在比特流中插入常量值以代替 CRC，且器件不会计算 CRC。 CRC 默认值为“Enable”，除非 BITSTREAM.ENCRYPTION.ENCRYPT 设为“Yes”，在此情况下禁用 CRC。

表 38：**7 系列比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No 和 Yes	此设置允许您创建调试比特流。调试比特流比标准比特流大得多。DebugBitstream 只能用于主串行配置和从串行配置。DebugBitstream 对于“Boundary Scan”（边界扫描）或者“Slave Parallel/SelectMAP”（从动并行/SelectMAP）无效。除了标准比特流外，调试比特流还可提供下列功能： 写入同步字后，将 32 个 0 写入 LOUT 寄存器。 单独加载每帧。 于每帧后执行循环冗余校验 (CRC)。 在每帧后，将帧地址写入 LOUT 寄存器。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No 和 Yes	此设置用于在完成配置后，禁用通过 JTAG 与 Boundary Scan (BSCAN) 块进行通信的功能。
BITSTREAM.GENERAL.JTAG_XADC	Enable	Enable、Disable 和 StatusOnly	此设置用于启用或禁用与 XADC 的 JTAG 连接。
BITSTREAM.GENERAL.PERFRAMECRC	No	No 和 Yes	在比特流中按固定间隔插入 CRC 值。这些值用于指示传入比特流的完整性，并且可在将配置数据加载到器件中之前标记错误（如 ICAP 的 INIT_B 管脚和 PRERROR 端口上所示）。虽然对于部分比特流而言，该属性设置为 Yes 最为合适，但它可将 CRC 值插入所有比特流，包括完整器件比特流。
BITSTREAM.GENERAL.XADCENHANCEDLINEARITY	Off	Off 和 On	此设置用于禁用部分内置数字校准功能，因为此类设置导致 INL 看上去比实际模拟性能更糟。
BITSTREAM.READBACK.ACTIVERECONFIG	No	No 和 Yes	此设置用于在配置期间，阻止断言 GHIGH 和 GSR 有效。这是动态部分重配置增强功能所必需的设置。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto、Top 和 Bottom	此设置用于选择顶部或底部 ICAP 端口。
BITSTREAM.READBACK.READBACK	False	True 和 False	此设置支持您通过创建必要的回读文件来执行回读功能。
BITSTREAM.READBACK.SECURITY	无	None、Level1 和 Level2	此设置用于指定是否禁用回读和重配置。 针对 Security 指定 Level1 即禁用回读。针对 Security 指定 Level2 即禁用回读和重配置。
BITSTREAM.READBACK.XADCPARTIALRECONFIG	Disable	Disable 和 Enable	禁用时，XADC 可在部分重配置期间持续工作。设为启用时，XADC 在部分重配置期间可在安全 (Safe) 模式下工作。
BITSTREAM.STARTUP.DONEPIPE	Yes	Yes 和 No	此设置用于告知 FPGA，等待 CFG_DONE (DONE) 管脚转至高电平，并等待首个时钟沿出现后，再转至 Done（完成）状态。
BITSTREAM.STARTUP.DONE_CYCLE	4	4、1、2、3、5、6 和 Keep	此设置用于选择激活 FPGA Done 信号的“Startup”（启动）阶段。当 DonePipe=Yes 时，则延迟转至“Done”（完成）状态。
BITSTREAM.STARTUP.GTS_CYCLE	5	5、1、2、3、4、6、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向 I/O 缓冲器释放内部三态控制。
BITSTREAM.STARTUP.GWE_CYCLE	6	6、1、2、3、4、5、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向触发器、LUT RAM 和移位寄存器断言内部写入使能有效。GWE_cycle 还会启用 BRAMS。在进入 Startup 阶段前，块 RAM 写入和读取都处于禁用状态。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait、0、1、2、3、4、5 和 6	此设置用于选择 Startup 阶段，在所选阶段内将等待至 DLL/DCM/PLL 锁定为止。如果选择 NoWait，则 Startup 顺序不会等待至 DLL/DCM/PLL 锁定。

表 38：**7 系列比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5 和 6	此设置用于指定在 Startup 周期内停滞，直至数控阻抗 (DCI) 匹配信号断言有效为止。DCI 匹配不会在 Match_cycle 上开始。Startup 序列会在此周期内等待至 DCI 匹配为止。鉴于判定 DCI 匹配所需时间过程中涉及多个变量，因此在任意给定系统中，完成 Startup 序列所需的 CCLK 周期数不尽相同。理想情况下，配置解决方案应持续驱动 CCLK 直至 DONE 转至“High”（高电平）为止。 当指定的设置为“Auto”（自动）时，`write_bitstream` 会搜索设计中是否包含任意 DCI I/O 标准。如果存在 DCI 标准，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=2。否则，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=NoWait。
BITSTREAM.STARTUP.STARTUPCLK	Cclk	Cclk、UserClk 和 JtagClk	器件配置后，StartupClk 序列可同步至 Cclk、用户时钟或 JTAG 时钟。默认值为 Cclk。 Cclk 允许您同步到 FPGA 器件中提供的内部时钟。 UserClk 允许您同步到用户定义的信号，此信号连接到 STARTUP 符号的 CLK 管脚。 JtagClk 允许您同步到 JTAG 提供的时钟。此时钟会对为 JTAG 提供控制逻辑的 TAP 控制器进行排序。 Spartan7 7s6 / 7s15 器件不支持 STARTUPE2.CLK (UserClk) 用户启动时钟管脚。

注释：

1. 对于专用配置管脚，AMD 建议您使用比特流默认设置。

Artix、Virtex 和 Kintex UltraScale+ 比特流设置

下表所示 AMD Artix™ UltraScale+™、AMD Virtex™ UltraScale+™ 和 AMD Kintex™ UltraScale+™ 器件的器件配置设置可搭配 `set_property <Setting> <Value> [current_design]` Vivado 工具 Tcl 命令一起使用。

表 39：**Artix UltraScale+、Virtex UltraScale+ 和 Kintex UltraScale+ 比特流设置**

设置	默认值	可能的值	描述
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	No 和 Yes	此设置用于指示是否使用 RSA 身份验证。如果设为 No，则使用 AES_GCM。
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE			此设置用于指定 OpenSSL .pem 文件，此文件包含的密钥对应用于签署经 RSA-2048 身份验证的比特流。
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3 或 4	此设置用于将 BPI 配置与闪存器件中的页面模式操作的时序进行同步。它允许您为首个页面的有效读取设置周期数。BPI_page_size 必须设置为 4 或 8，这样该选项才可用。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	对于 BPI 配置，该子选项允许您指定页面大小，此大小对应于闪存每个页面所需读取数。
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1 和 Type2	此设置用于为不同类型的 BPI 闪存器件设置 BPI 同步配置模式。 Disable（默认值）即禁用同步配置模式。 Type1 启用同步配置模式和设置以支持 Micron G18(F) 系列。 Type2 启用同步配置模式和设置以支持 Micron (Numonyx) P30 和 P33 系列。

表 39：**Artix UltraScale+、Virtex UltraScale+ 和 Kintex UltraScale+ 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.CCLKPIN	Pullup	Pullup 和 Pullnone	此设置用于向 Cclk 管脚添加内部上拉。Pullnone 设置则禁用上拉。
BITSTREAM.CONFIG.PERSIST	No	No 和 Yes	禁止使用配置管脚作为用户 I/O，且配置后仍保留此设置。
BITSTREAM.CONFIG.CONFIGRATE	2.7	2.7、5.3、8.0、10.6、21.3、31.9、36.4、51.0、56.7、63.8、72.9、85.0、102.0、127.5、170.0	在主模式下配置时，比特流生成使用内部振荡器来生成配置时钟 Cclk。该子选项用于选择 Cclk 的速率。
BITSTREAM.CONFIG.D00_MOSI	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D00_MOSI 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D00_MOSI 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.D01_DIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D01_DIN 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D01_DIN 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.D02	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D02 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D02 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.D03	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D03 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D03 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Quiet 和 Safe	此设置用于控制数控阻抗电路尝试对 DCI IOSTANDARD 执行阻抗匹配更新的频率。
BITSTREAM.CONFIG.DONEPIN	Pullup	Pullup 和 Pullnone	此设置用于向 DONE 管脚添加内部上拉。Pullnone 设置则禁用上拉。仅当您要将外部上拉电阻器连接到此管脚时，才使用 DonePin。如果您不使用 DonePin，则自动连接内部上拉电阻器。
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable、Div-1、Div-2 和 Div-3。Div-4、Div-6、Div-8、Div-12、Div-16、Div-24 和 Div-48	此设置允许使用外部时钟作为所有主模式的配置时钟。此外部时钟必须连接到两用 EMCCLK 管脚。
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH	无	指向 familyKey.cfg 的路径	此设置用于指定族密钥的安装位置。无需特定目录。 AMD 并未在其 AMD 工具套件中提供族密钥。客户如需族密钥，必须向 secure.solutions@xilinx.com 发送请求。将通过 https://china.xilinx.com 上的“产品许可”站点向合格客户分配族密钥。
BITSTREAM.CONFIG.INITPIN	Pullup	Pullup 和 Pullnone	此设置用于指定是要将 Pullup 电阻器添加到 INIT 管脚，还是保留 INIT 管脚处于浮动状态。
BITSTREAM.CONFIG.M0PIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M0 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M0 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M1PIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M1 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M1 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M2PIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M2 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M2 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	无	<字符串>	此设置用于在 MultiBoot 设置（存储在 WBSTAR 寄存器中）中设置下一次配置的起始地址。

表 39：**Artix UltraScale+、Virtex UltraScale+ 和 Kintex UltraScale+ 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable 和 Disable	设置为 Disable 时，将从 `.bit` 文件中移除 IPROG 命令。
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable 和 Disable	此设置用于启用或禁用 SelectMAP 模式“Abort”（异常中止）序列。如果禁用，则忽略器件管脚上的 Abort 序列。
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Enable 和 Disable	当配置尝试失败时，启用或禁用默认比特流的加载。
BITSTREAM.CONFIG.PROGPIN	Pullup	Pullup 和 Pullnone	此设置用于向 PROGRAM_B 管脚添加内部上拉。Pullnone 设置则禁用上拉。配置 Pullup 后，上拉会影响管脚。
BITSTREAM.CONFIG.PUDC_B	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 PUDC_B 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 PUDC_B 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.RDWR_B_FCS_B	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 RDWR_B_FCS_B 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 RDWR_B_FCS_B 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.REVISIONSELECT	00	00、01、10 或 11	此设置用于在温态启动起始地址 (WBSTAR) 寄存器中为下一次温态启动指定 RS[1:0] 设置的内部值。
BITSTREAM.CONFIG.REVISIONSELECT_ TRISTATE	Disable	Disable 和 Enable	此设置用于通过在温态启动起始地址 (WBSTAR) 中设置相应选项来指定是否启用 RS[1:0] 三态。 RS[1:0] 管脚三态启用 0：启用 RS 三态 1：禁用 RS 三态
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable 和 Enable	此设置用于在“System Monitor”（系统监控器）检测到温度超出可接受的最大工作温度时关闭器件。需为系统监控器设置外部电路才能使用该选项。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No 和 Yes	此设置启用 SPI 32 位地址样式，对于存储空间不小于 256 Mb 的 SPI 器件，此设置是必需的。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2、4 和 8	针对来自第三方 SPI 闪存器件的“Master SPI”（主 SPI）配置，将 SPI 总线设置为“Dual (x2)”（双通道）或“Quad (x4)”（四通道）模式。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No 和 Yes	此设置用于将 FPGA 设置为使用下降沿时钟进行 SPI 数据捕获。这样可改进时序裕度，并能够提升配置的时钟速率。
BITSTREAM.CONFIG.TCKPIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TCK 管脚添加上拉、下拉或两者都不添加，此管脚是 JTAG 测试时钟。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDIPIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDI 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和 JTAG 寄存器的串行数据输入。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDOPIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDO 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和数据寄存器的串行数据输出。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TIMER_CFG			在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_USR 同时使用。
BITSTREAM.CONFIG.TIMER_USR			在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_CFG 同时使用。
BITSTREAM.CONFIG.TMSPIN	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TMS 管脚添加上拉、下拉或两者都不添加，此管脚是 TAP 控制器的模式输入信号。TAP 控制器可为 JTAG 提供控制逻辑。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。

表 39：**Artix UltraScale+、Virtex UltraScale+ 和 Kintex UltraScale+ 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pullup、Pulldown 和 Pullnone	此设置用于向未使用的 SelectIO™ 管脚 (IOB) 添加上拉、下拉或者两者都不添加。它对于专用配置管脚无效。专用配置管脚列表因架构而异。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	0xFFFFFFFF	此设置用于识别实现的版本。您可在“User ID”（用户 ID）寄存器中添加最多 1 个含 8 个数字的十六进制字符串。
BITSTREAM.CONFIG.USR_ACCESS	无	None、<8 个数字的十六进制字符串> 和 TIMESTAMP	此设置用于将 1 个含 8 个数字的十六进制字符串或时间戳写入 AXSS 配置寄存器。时间戳值的格式为 ddddd MMMM yyyyyy hhhhh mmmmmm ssssss：对应日、月、年（2000 年 = 00000）、小时、分钟、秒。FPGA 互连结构可通过 USR_ACCESS 原语直接访问此寄存器的内容。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable 和 Disable	设为“Enabled”（启用）后，如果检测到配置错误，则 INIT_B 管脚断言为“0”。
BITSTREAM.ENCRYPTION.ENCRYPT	No	No 和 Yes	加密比特流。
BITSTREAM.ENCRYPTION.DEBUGKDFKEYS	No	No 和 Yes	设为启用后，即可生成包含 KDF 模式下生成的所有密钥的调试文件。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram 和 efuse	此设置用于判断将使用的 AES 加密密钥的位置：密钥来自于电池供电式 RAM (BBRAM) 或 eFUSE 寄存器。仅当“Encrypt”选项设置为“True”时，该属性才可用。
BITSTREAM.ENCRYPTION.OBFUSCATEKEY	Disable	Disable 和 Enable	当 AES 密钥无读保护时，读取密钥会返回密钥的 CRC 散列，而不是实际密钥值。
BITSTREAM.ENCRYPTION.KEY0			Key0 用于为比特流加密设置 64 位 AES 加密密钥。要将此项设置为可选，请将此生成器留空以便选择随机数值。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.STARTIV0			用于设置 AES 初始矢量起始值。仅将该 128 位值的前 96 位用于初始化矢量。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE			用于设置 128 位模糊初始矢量起始值。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.KDFFIXEDINPUT	无		可选 60 字节固定输入值，指定为 120 个数字的十六进制值。此 60 字节固定输入搭配 4 字节计数器即可充当 KDF 的虚拟随机函数 (PRF) 的 64 字节固定输入数据，用于生成 32 字节密钥输出 (KO)。如不指定此设置，则 write_bitstream 会使用 RAND_bytes 生成 60 字节虚拟随机固定输入值。
BITSTREAM.ENCRYPTION.KDFSEED	无		可选 32 字节 KDF 种子值，指定为 64 个数字的十六进制值。如不指定此设置，则 write_bitstream 会从输入 .nky 文件中取 Key0 值作为种子值。如果不指定此设置，并且不存在来自 NKY 文件的 Key0 输入，那么 write_bitstream 会通过 RAND_bytes 生成 32 字节虚拟随机种子值。
BITSTREAM.ENCRYPTION.KEYFILE			此设置用于指定输入加密文件的名称（含 .nky 文件扩展名）。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.KEYLIFE	32	4 到 2147483647	128 位加密块的数量，单一密钥应基于此类加密块并用于经 AES-GCM 身份验证的比特流。
BITSTREAM.ENCRYPTION.RSAKEYLIFEFRAMES	8	8 到 2147483647	在已指定使用 RSA 公钥进行身份验证的情况下，此设置用于指定应用于任意给定 AES-256 密钥的配置帧的数量。配置帧数为 8 等同于使用含 246 个加密块的密钥。
BITSTREAM.GENERAL.COMPRESS	False	True 和 False	比特流中的多帧写入功能用于减小比特流的大小，而不是减小比特文件的大小。使用 COMPRESS 并不保证比特流大小会缩小。

表 39：**Artix UltraScale+、Virtex UltraScale+ 和 Kintex UltraScale+ 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.GENERAL.CRC	Enable	Enable 和 Disable	此设置用于控制比特流中循环冗余校验 (CRC) 值的生成。设为启用后，可根据比特流内容计算出唯一的 CRC 值。如果计算所得的 CRC 值与比特流中的 CRC 值不匹配，那么器件将无法进行配置。禁用 CRC 时，将在比特流中插入常量值以代替 CRC，且器件不会计算 CRC。 CRC 默认值为“Enable”，除非 BITSTREAM.ENCRYPTION.ENCRYPT 设为“Yes”，在此情况下禁用 CRC。
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No 和 Yes	此设置允许您创建调试比特流。调试比特流比标准比特流大得多。DebugBitstream 只能用于主串行配置和从串行配置。DebugBitstream 对于“Boundary Scan”（边界扫描）或者“Slave Parallel/SelectMAP”（从动并行/SelectMAP）无效。除了标准比特流外，调试比特流还可提供下列功能：写入同步字后，将 32 个 0 写入 LOUT 寄存器。单独加载每帧。于每帧后执行循环冗余校验 (CRC)。在每帧后，将帧地址写入 LOUT 寄存器。
BITSTREAM.GENERAL.PERFRAMECRC	No	No 和 Yes	在比特流中按固定间隔插入 CRC 值。这些值用于指示传入比特流的完整性，并且可在将配置数据加载到器件中之前标记错误（如 ICAP 的 INIT_B 管脚和 PRERROR 端口上所示）。虽然对于部分比特流而言，该属性设置为 Yes 最为合适，但它可将 CRC 值插入所有比特流，包括完整器件比特流。
BITSTREAM.GENERAL.SYSMONPOWERDOWN	Disable	Disable 和 Enable	设为启用后，即可支持器件将 SYSMON 下电，以节省功耗。建议此设置仅用于将 SYSMON 永久下电。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No 和 Yes	此设置用于在完成配置后，禁用通过 JTAG 与 Boundary Scan (BSCAN) 块进行通信的功能。
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable、Disable 和 StatusOnly	此设置用于启用或禁用与 SYSMON 的 JTAG 连接。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto、Top 和 Bottom	此设置用于选择顶部或底部 ICAP 端口。
BITSTREAM.READBACK.ACTIVERECONFIG	No	No 和 Yes	此设置用于在配置期间，阻止断言 GHIGH 和 GSR 有效。这是动态部分重配置增强功能所必需的设置。
BITSTREAM.READBACK.SECURITY	无	None、Level1 和 Level2	此设置用于指定是否禁用回读和重配置。 针对 Security 指定 Level1 即禁用回读。
BITSTREAM.STARTUP.DONE_CYCLE	4	4、1、2、3、5 和 6	此设置用于选择激活 FPGA Done 信号的“Startup”（启动）阶段。当 DonePipe=Yes 时，则延迟转至“Done”（完成）状态。
BITSTREAM.STARTUP.GTS_CYCLE	5	5、1、2、3、4、6、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向 I/O 缓冲器释放内部三态控制。
BITSTREAM.STARTUP.GWE_CYCLE	6	6、1、2、3、4、5、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向触发器、LUT RAM 和移位寄存器断言内部写入使能有效。GWE_cycle 还会启用 BRAMS。在进入 Startup 阶段前，块 RAM 写入和读取都处于禁用状态。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait、0、1、2、3、4、5 和 6	此设置用于选择 Startup 阶段，在所选阶段内将等待至 MMCM/PLL 锁定为止。如果选择 NoWait，则 Startup 顺序不会等待至 MMCM/PLL 锁定。

表 39：**Artix UltraScale+、Virtex UltraScale+ 和 Kintex UltraScale+ 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5 和 6	此设置用于指定在 Startup 周期内停滞，直至数控阻抗 (DCI) 匹配信号断言有效为止。DCI 匹配不会在 Match_cycle 上开始。Startup 序列会在此周期内等待至 DCI 匹配为止。鉴于判定 DCI 匹配所需时间过程中涉及多个变量，因此在任意给定系统中，完成 Startup 序列所需的 CCLK 周期数不尽相同。理想情况下，配置解决方案应持续驱动 CCLK 直至 DONE 转至“High”（高电平）为止。 当指定的设置为“Auto”（自动）时，`write_bitstream` 会搜索设计中是否包含任意 DCI I/O 标准。如果存在 DCI 标准，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=2。否则，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=NoWait。

UltraScale 比特流设置

下表所示 AMD UltraScale™ 器件的器件配置设置可搭配 `set_property <Setting> <Value> [current_design]` Vivado 工具 Tcl 命令一起使用。

表 40：**UltraScale 比特流设置**

设置	默认值	可能的值	描述
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	Yes 和 No	此设置用于指示是否使用 RSA 身份验证。如果设为 No，则使用 AES_GCM。
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE	无	<字符串>	此设置用于指定 OpenSSL .pem 文件，此文件包含的密钥对应用于签署经 RSA-2048 身份验证的比特流。
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3 或 4	此设置用于将 BPI 配置与闪存器件中的页面模式操作的时序进行同步。它允许您为首个页面的有效读取设置周期数。BPI_page_size 必须设置为 4 或 8，这样该选项才可用。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	对于 BPI 配置，该选项允许您指定页面大小，此大小对应于闪存每个页面所需读取数。
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1 和 Type2	此设置用于为不同类型的 BPI 闪存器件设置 BPI 同步配置模式。 Disable（默认值）即禁用同步配置模式。 Type1 启用同步配置模式和设置以支持 Micron G18(F) 系列。 Type2 启用同步配置模式和设置以支持 Micron (Numonyx) P30 和 P33 系列。
BITSTREAM.CONFIG.CCLKPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于向 Cclk 管脚添加内部上拉。Pullnone 设置则禁用上拉。
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Disable 和 Enable	当配置尝试失败时，启用或禁用默认比特流的加载。
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、22、33、40、50、57、69、82、87、90、110、115、130 和 148	在主模式下配置时，使用内部振荡器来生成配置时钟 Cclk。该选项用于选择 Cclk 的速率。
BITSTREAM.CONFIG.D00_MOSI[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D00_MOSI 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D00_MOSI 管脚上的上拉电阻器和下拉电阻器。

表 40：**UltraScale 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.D01_DIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D01_DIN 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D01_DIN 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.D02[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D02 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D02 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.D03[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 D03 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 D03 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Continuous 和 Quiet	此设置用于控制数控阻抗电路尝试对 DCI IOSTANDARD 执行阻抗匹配更新的频率。
BITSTREAM.CONFIG.DONEPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于向 DONE 管脚添加内部上拉。Pullnone 设置则禁用上拉。仅当您要将外部上拉电阻器连接到此管脚时，才使用 DonePin。如果您不使用 DonePin，则自动连接内部上拉电阻器。
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable、Div-1、Div-2、Div-3、Div-4、Div-6、Div-8、Div-12、Div-16、Div-24 和 Div-48	此设置允许使用外部时钟作为所有主模式的配置时钟。此外部时钟必须连接到两用 EMCCLK 管脚。
BITSTREAM.CONFIG.INITPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于指定是要将 Pullup 电阻器添加到 INIT 管脚，还是保留 INIT 管脚处于浮动状态。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable 和 Disable	设为“Enabled”（启用）后，如果检测到配置错误，则 INIT_B 管脚断言为“0”。
BITSTREAM.CONFIG.M0PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M0 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M0 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M1PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M1 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M1 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M2PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M2 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M2 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	无	<字符串>	此设置用于在 MultiBoot 设置（存储在 WBSTAR 寄存器中）中设置下一次配置的起始地址。
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable 和 Disable	设置为 Disable 时，将从 `.bit` 文件中移除 IPROG 命令。这样即可允许在上电时加载黄金镜像，而不是跳转到多重启动配置中的多重启动镜像。
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable 和 Enable	此设置用于在“System Monitor”（系统监控器）检测到温度超出可接受的最大工作温度时关闭器件。需为系统监控器设置外部电路才能使用该选项。
BITSTREAM.CONFIG.PERSIST	No	No 和 Yes	此设置用于在完成配置后，保留对多功能配置管脚的配置逻辑访问权。此设置主要用于在配置后保留 SelectMAP 端口用于回读访问，但也可配合任意配置模式使用。对于 JTAG 配置则无需保留 (PERSIST)，因为 JTAG 端口是专用端口且始终可用。PERSIST 和 ICAP 不能同时使用。 请参阅用户指南以获取相关说明。针对使用 SelectMAP 配置管脚的“Readback”（回读）和“Partial Reconfiguration”（部分重配置）操作，需使用 PERSIST，并且使用 SelectMAP 模式或串行模式时，也应使用 PERSIST。

表 40：**UltraScale 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.PROGPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于向 PROGRAM_B 管脚添加内部上拉。Pullnone 设置则禁用上拉。配置 Pullup 后，上拉会影响管脚。
BITSTREAM.CONFIG.PUDC_B[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 PUDC_B 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 PUDC_B 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.RDWR_B_FCS_B[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 RDWR_B_FCS_B 管脚添加内部上拉、下拉或者两者都不添加。 选择 Pullnone 以禁用 RDWR_B_FCS_B 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.REVISIONSELECT	00	00、01、10 或 11	此设置用于在温态启动起始地址 (WBSTAR) 寄存器中为下一次温态启动指定 RS[1:0] 设置的内部值。
BITSTREAM.CONFIG.REVISIONSELECT_ TRISTATE	Disable	Disable 和 Enable	此设置用于通过在温态启动起始地址 (WBSTAR) 中设置相应选项来指定是否启用 RS[1:0] 三态。 RS[1:0] 管脚三态启用 0：启用 RS 三态 1：禁用 RS 三态
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable 和 Disable	此设置用于启用或禁用 SelectMAP 模式“Abort”（异常中止）序列。如果禁用，则忽略器件管脚上的 Abort 序列。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No 和 Yes	此设置启用 SPI 32 位地址样式，对于存储空间不小于 256 Mb 的 SPI 器件，此设置是必需的。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2、4 和 8	针对来自第三方 SPI 闪存器件的“Master SPI”（主 SPI）配置，将 SPI 总线设置为“Dual (x2)”（双通道）或“Quad (x4)”（四通道）模式。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No 和 Yes	此设置用于将 FPGA 设置为使用下降沿时钟进行 SPI 数据捕获。这样可改进时序裕度，并能够提升配置的时钟速率。
BITSTREAM.CONFIG.TCKPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TCK 管脚添加上拉、下拉或两者都不添加，此管脚是 JTAG 测试时钟。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDIPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDI 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和 JTAG 寄存器的串行数据输入。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDOPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDO 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和数据寄存器的串行数据输出。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TIMER_CFG	无	<8 个数字的十六进制字符串>	在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_USR 同时使用。
BITSTREAM.CONFIG.TIMER_USR	0x00000000	<8 个数字的十六进制字符串>	在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_CFG 同时使用。
BITSTREAM.CONFIG.TMSPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TMS 管脚添加上拉、下拉或两者都不添加，此管脚是 TAP 控制器的模式输入信号。TAP 控制器可为 JTAG 提供控制逻辑。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown、Pullup 和 Pullnone	此设置用于向未使用的 SelectIO 管脚 (IOB) 添加上拉、下拉或者两者都不添加。它对于专用配置管脚无效。专用配置管脚列表因架构而异。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	<8 个数字的十六进制字符串>	此设置用于识别实现的版本。您可在“User ID”（用户 ID）寄存器中添加最多 1 个含 8 个数字的十六进制字符串。

表 40：**UltraScale 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.USR_ACCESS	无	<8 个数字的十六进制字符串> 和 TIMESTAMP	此设置用于将 1 个含 8 个数字的十六进制字符串或时间戳写入 AXSS 配置寄存器。时间戳值的格式为 ddddd MMMM yyyyyy hhhhh mmmmmm ssssss：对应日、月、年（2000 年 = 00000）、小时、分钟、秒。FPGA 互连结构可通过 USR_ACCESS 原语直接访问此寄存器的内容。
BITSTREAM.ENCRYPTION.ENCRYPT	No	No 和 Yes	加密比特流。
BITSTREAM.ENCRYPTION.DEBUGKDFKEYS	No	No 和 Yes	设为启用后，即可生成包含 KDF 模式下生成的所有密钥的调试文件。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram 和 efuse	此设置用于判断将使用的 AES 加密密钥的位置：密钥来自于电池供电式 RAM (BBRAM) 或 eFUSE 寄存器。 仅当“Encrypt”选项设置为“True”时，该属性才可用。
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH	无	指向 familyKey.cfg 的路径	此设置用于指定族密钥的安装位置。无需特定目录。 AMD 并未在其 AMD 工具套件中提供族密钥。客户如需族密钥，必须向 secure.solutions@xilinx.com 发送请求。将通过 https://china.xilinx.com 上的“产品许可”站点向合格客户分配族密钥。
BITSTREAM.ENCRYPTION.KEY0	无	<十六进制字符串>	Key0 用于为比特流加密设置 AES 加密密钥。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.KEYFILE	无	<字符串>	此设置用于指定输入加密文件的名称（含 `.nky` 文件扩展名）。要使用该选项，必须首先将“Encrypt”设置为“Yes”。
BITSTREAM.ENCRYPTION.KEYLIFE	32	4 到 2147483647	128 位加密块的数量，单一密钥应基于此类加密块并用于经 AES-GCM 身份验证的比特流。 将此项设为 0 即可禁用这些选项
BITSTREAM.ENCRYPTION.RSAKEYLIFEFRAMES	8	8 到 2147483647	在已指定使用 RSA 公钥进行身份验证的情况下，此设置用于指定应用于任意给定 AES-256 密钥的配置帧的数量。配置帧数为 8 等同于使用含 246 个加密块的密钥。 将此项设为 0 即可禁用这些选项。
BITSTREAM.ENCRYPTION.OBFUSCATEKEY	Disable	Enable 和 Disable	创建比特流，其中用于加密比特流的密钥将先转换为模糊密钥，然后再写入 eFUSE 或电池供电式 RAM (BBR)。这样即可允许用户为器件烧录器提供模糊密钥，而不是原始客户密钥。器件烧录器可将模糊密钥写入 eFUSE 或 BBR 中，并使用所选存储位置中的 obfuscated-key 标志将其标记为模糊密钥。
BITSTREAM.ENCRYPTION.STARTIV0			此设置用于指定第一条 AES-GCM 消息中的初始 GCM 计数值的初始化矢量。128 位十六进制值。
BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE			用于设置 AES 初始矢量起始值。仅将该 128 位值的前 96 位用于初始化矢量。要使用该选项，必须首先将“Encrypt”设置为“Yes”
BITSTREAM.ENCRYPTION.KDFFIXEDINPUT	无		可选 60 字节固定输入值，指定为 120 个数字的十六进制值。此 60 字节固定输入搭配 4 字节计数器即可充当 KDF 的虚拟随机函数 (PRF) 的 64 字节固定输入数据，用于生成 32 字节密钥输出 (KO)。如不指定此设置，则 write_bitstream 会通过 RAND_bytes 生成 60 字节虚拟随机固定输入值。
BITSTREAM.ENCRYPTION.KDFSEED	无		可选 32 字节 KDF 种子值，指定为 64 个数字的十六进制值。如不指定此设置，则 write_bitstream 会从输入 .nky 文件中取 Key0 值作为种子值。如果不指定此设置，并且不存在来自 NKY 文件的 Key0 输入，那么 write_bitstream 会通过 RAND_bytes 生成 32 字节虚拟随机种子值。
BITSTREAM.GENERAL.COMPRESS	False	True 和 False	比特流中的多帧写入功能用于减小比特流的大小，而不是用于减小比特流 (`.bit`) 文件的大小。使用 COMPRESS 并不保证比特流大小会缩小。

表 40：**UltraScale 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.GENERAL.CRC	Enable	Enable 和 Disable	此设置用于控制比特流中循环冗余校验 (CRC) 值的生成。设为启用后，可根据比特流内容计算出唯一的 CRC 值。如果计算所得的 CRC 值与比特流中的 CRC 值不匹配，那么器件将无法进行配置。禁用 CRC 时，将在比特流中插入常量值以代替 CRC，且器件不会计算 CRC。 CRC 默认值为“Enable”，除非 BITSTREAM.ENCRYPTION.ENCRYPT 设为“Yes”，在此情况下禁用 CRC。
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No 和 Yes	此设置允许您创建调试比特流。调试比特流比标准比特流大得多。DebugBitstream 只能用于主串行配置和从串行配置。DebugBitstream 对于“Boundary Scan”（边界扫描）或者“Slave Parallel/SelectMAP”（从动并行/SelectMAP）无效。除了标准比特流外，调试比特流还可提供下列功能： 写入同步字后，将 32 个 0 写入 LOUT 寄存器。 单独加载每帧。 于每帧后执行循环冗余校验 (CRC)。 在每帧后，将帧地址写入 LOUT 寄存器。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No 和 Yes	此设置用于在完成配置后，禁用通过 JTAG 与 Boundary Scan (BSCAN) 块进行通信的功能。
BITSTREAM.GENERAL.PERFRAMECRC	No	No 和 Yes	在比特流中按固定间隔插入 CRC 值。这些值用于指示传入比特流的完整性，并且可在将配置数据加载到器件中之前标记错误（如 ICAP 的 INIT_B 管脚和 PRERROR 端口上所示）。虽然对于部分比特流而言，该属性设置为 Yes 最为合适，但它可将 CRC 值插入所有比特流，包括完整器件比特流。
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable、Disable 和 StatusOnly	此设置用于启用或禁用与 SYSMON 的 JTAG 连接。
BITSTREAM.GENERAL.SYSMONPOWERDOWN	Disable	Disable 和 Enable	设为启用后，即可支持器件将 SYSMON 下电，以节省功耗。建议此设置仅用于将 SYSMON 永久下电。
BITSTREAM.MMCM.BANDWIDTH	DEFAULT	POSTCRC	通过将 BANDWIDTH 设置从 OPTIMIZED 更改为 POSTCRC 来更改所有 MMCM。
BITSTREAM.PLL.BANDWIDTH	DEFAULT	POSTCRC	通过将 BANDWIDTH 设置从 OPTIMIZED 更改为 POSTCRC 来更改所有 PLL。
BITSTREAM.READBACK.ACTIVERECONFIG	No	No 和 Yes	此设置用于在配置期间，阻止断言 GHIGH 和 GSR 有效。这是动态部分重配置增强功能所必需的设置。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto、Top 和 Bottom	此设置用于选择顶部或底部 ICAP 端口。
BITSTREAM.READBACK.READBACK	False	True 和 False	此设置支持您通过创建必要的回读文件来执行回读功能。
BITSTREAM.READBACK.SECURITY	无	None、Level1 和 Level2	此设置用于指定是否禁用回读和重配置。 针对 Security 指定 Level1 即禁用回读。针对 Security 指定 Level2 即禁用回读和重配置。
BITSTREAM.STARTUP.DONE_CYCLE	4	4、1、2、3、5、6 和 Keep	此设置用于选择激活 FPGA Done 信号的“Startup”（启动）阶段。当 DonePipe=Yes 时，则延迟转至“Done”（完成）状态。
BITSTREAM.STARTUP.GTS_CYCLE	5	5、1、2、3、4、6、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向 I/O 缓冲器释放内部三态控制。
BITSTREAM.STARTUP.GWE_CYCLE	6	6、1、2、3、4、5、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向触发器、LUT RAM 和移位寄存器断言内部写入使能有效。GWE_cycle 还会启用 BRAMS。在进入 Startup 阶段前，块 RAM 写入和读取都处于禁用状态。

表 40：**UltraScale 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait、0、1、2、3、4、5 和 6	此设置用于选择 Startup 阶段，在所选阶段内将等待至 DLL/DCM/PLL 锁定为止。如果选择 NoWait，则 Startup 顺序不会等待至 DLL/DCM/PLL 锁定。
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5 和 6	此设置用于指定在 Startup 周期内停滞，直至数控阻抗 (DCI) 匹配信号断言有效为止。DCI 匹配不会在 Match_cycle 上开始。Startup 序列会在此周期内等待至 DCI 匹配为止。鉴于判定 DCI 匹配所需时间过程中涉及多个变量，因此在任意给定系统中，完成 Startup 序列所需的 CCLK 周期数不尽相同。理想情况下，配置解决方案应持续驱动 CCLK 直至 DONE 转至“High”（高电平）为止。 当指定的设置为“Auto”（自动）时，`write_bitstream` 会搜索设计中是否包含任意 DCI I/O 标准。如果存在 DCI 标准，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=2。否则，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=NoWait。

注释：

1. 对于专用配置管脚，AMD 建议您使用默认比特流设置。

Zynq UltraScale+ MPSoC 比特流设置

下表所示 AMD Zynq™ UltraScale+™ MPSoC 器件的器件配置设置可搭配 `set_property <Setting> <Value> [current_design]` Vivado 工具 Tcl 命令一起使用。

表 41：**Zynq UltraScale+ MPSoC 比特流设置**

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Quiet 和 Safe	此设置用于控制数控阻抗电路尝试对 DCI IOSTANDARD 执行阻抗匹配更新的频率。
BITSTREAM.CONFIG.PUDC_B	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 PUDC_B 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 PUDC_B 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable 和 Enable	此设置用于在“System Monitor”（系统监控器）检测到温度超出可接受的最大工作温度时关闭器件。需为系统监控器设置外部电路才能使用该选项。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pullup、Pulldown 和 Pullnone	此设置用于向未使用的 SelectIO 管脚 (IOB) 添加上拉、下拉或者两者都不添加。它对于专用配置管脚无效。专用配置管脚列表因架构而异。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	0xFFFFFFFF	此设置用于识别实现的版本。您可在“User ID”（用户 ID）寄存器中添加最多 1 个含 8 个数字的十六进制字符串。
BITSTREAM.CONFIG.USR_ACCESS	无	None、<8 个数字的十六进制字符串> 和 TIMESTAMP	此设置用于将 1 个含 8 个数字的十六进制字符串或时间戳写入 AXSS 配置寄存器。时间戳值的格式为 ddddd MMMM yyyyyy hhhhh mmmmmm ssssss：对应日、月、年（2000 年 = 00000）、小时、分钟、秒。FPGA 互连结构可通过 USR_ACCESS 原语直接访问此寄存器的内容。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable 和 Disable	设为“Enabled”（启用）后，如果检测到配置错误，则 INIT_B 管脚断言为“0”。

表 41：**Zynq UltraScale+ MPSoC 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.GENERAL.COMPRESS	False	True 和 False	比特流中的多帧写入功能用于减小比特流的大小，而不是减小比特文件的大小。使用 COMPRESS 并不保证比特流大小会缩小。
BITSTREAM.GENERAL.CRC	Enable	Enable 和 Disable	此设置用于控制比特流中循环冗余校验 (CRC) 值的生成。设为启用后，可根据比特流内容计算出唯一的 CRC 值。如果计算所得的 CRC 值与比特流中的 CRC 值不匹配，那么器件将无法进行配置。禁用 CRC 时，将在比特流中插入常量值以代替 CRC，且器件不会计算 CRC。
BITSTREAM.GENERAL.PERFRAMECRC	No	No 和 Yes	在比特流中按固定间隔插入 CRC 值。这些值用于指示传入比特流的完整性，并且可在将配置数据加载到器件中之前标记错误（如 ICAP 的 INIT_B 管脚和 PRERROR 端口上所示）。虽然对于部分比特流而言，该属性设置为 Yes 最为合适，但它可将 CRC 值插入所有比特流，包括完整器件比特流。
BITSTREAM.GENERAL.SYSMONPOWERDOWN	Disable	Disable 和 Enable	设为启用后，即可支持器件将 SYSMON 下电，以节省功耗。建议此设置仅用于将 SYSMON 永久下电。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No 和 Yes	此设置用于在完成配置后，禁用通过 JTAG 与 Boundary Scan (BSCAN) 块进行通信的功能。
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable、Disable 和 StatusOnly	此设置用于启用或禁用与 SYSMON 的 JTAG 连接。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto、Top 和 Bottom	此设置用于选择顶部或底部 ICAP 端口。
BITSTREAM.READBACK.ACTIVERECONFIG	No	No 和 Yes	此设置用于在配置期间，阻止断言 GHIGH 和 GSR 有效。这是动态部分重配置增强功能所必需的设置。
BITSTREAM.READBACK.SECURITY	无	None、Level1 和 Level2	此设置用于指定是否禁用回读和重配置。 针对 Security 指定 Level1 即禁用回读。针对 Security 指定 Level2 则禁用回读和重配置。
BITSTREAM.STARTUP.DONE_CYCLE	4	4、1、2、3、5、6 和 Keep	此设置用于选择激活 FPGA Done 信号的“Startup”（启动）阶段。当 DonePipe=Yes 时，则延迟转至 Done 状态
BITSTREAM.STARTUP.GTS_CYCLE	5	5、1、2、3、4、6、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向 I/O 缓冲器释放内部三态控制
BITSTREAM.STARTUP.GWE_CYCLE	6	6、1、2、3、4、5、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向触发器、LUT RAM 和移位寄存器断言内部写入使能有效。GWE_cycle 还会启用 BRAMS。在进入 Startup 阶段前，块 RAM 写入和读取都处于禁用状态。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait、0、1、2、3、4、5 和 6	此设置用于选择 Startup 阶段，在所选阶段内将等待至 MMCM/PLL 锁定为止。如果选择 NoWait，则 Startup 顺序不会等待至 MMCM/PLL 锁定。
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5 和 6	此设置用于指定在 Startup 周期内停滞，直至数控阻抗 (DCI) 匹配信号断言有效为止。DCI 匹配不会在 Match_cycle 上开始。Startup 序列会在此周期内等待至 DCI 匹配为止。鉴于判定 DCI 匹配所需时间过程中涉及多个变量，因此在任意给定系统中，完成 Startup 序列所需的 CCLK 周期数不尽相同。理想情况下，配置解决方案应持续驱动 CCLK 直至 DONE 转至“High”（高电平）为止。 当指定的设置为“Auto”（自动）时，`write_bitstream` 会搜索设计中是否包含任意 DCI I/O 标准。如果存在 DCI 标准，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=2。否则，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=NoWait。

Zynq 7000 比特流设置

下表所示 AMD Zynq™ 7000 器件的器件配置设置可搭配 `set_property <Setting> <Value> [current_design]` Vivado 工具 Tcl 命令一起使用。

注释：用于加密的比特流设置对 Zynq 7000 器件无效。

表 42：**Zynq 7000 比特流设置**

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3 或 4	此设置用于将 BPI 配置与闪存器件中的页面模式操作的时序进行同步。它允许您为首个页面的有效读取设置周期数。BPI_page_size 必须设置为 4 或 8，这样该选项才可用。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	对于 BPI 配置，该选项允许您指定页面大小，此大小对应于闪存每个页面所需读取数。
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1 和 Type2	此设置用于为不同类型的 BPI 闪存器件设置 BPI 同步配置模式。 Disable（默认值）即禁用同步配置模式。 Type1 启用同步配置模式和设置以支持 Micron G18(F) 系列。 Type2 启用同步配置模式和设置以支持 Micron (Numonyx) P30 和 P33 系列。
BITSTREAM.CONFIG.CCLKPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于向 Cclk 管脚添加内部上拉。Pullnone 设置则禁用上拉。
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Disable 和 Enable	当配置尝试失败时，启用或禁用默认比特流的加载。
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、16、22、26、33、40、50 和 66	在主模式下配置时，使用内部振荡器来生成配置时钟 Cclk。该选项用于选择 Cclk 的速率。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Continuous 和 Quiet	此设置用于控制数控阻抗电路尝试对 DCI IOSTANDARD 执行阻抗匹配更新的频率。
BITSTREAM.CONFIG.DONEPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于向 DONE 管脚添加内部上拉。Pullnone 设置则禁用上拉。仅当您要将外部上拉电阻器连接到此管脚时，才使用 DonePin。如果您不使用 DonePin，则自动连接内部上拉电阻器。
BITSTREAM.CONFIG.INITPIN[1]	Pullup	Pullup 和 Pullnone	此设置用于指定是要将 Pullup 电阻器添加到 INIT 管脚，还是保留 INIT 管脚处于浮动状态。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable 和 Disable	设为“Enabled”（启用）后，如果检测到配置错误，则 INIT_B 管脚断言为“0”。
BITSTREAM.CONFIG.M0PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M0 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M0 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M1PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M1 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M1 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.M2PIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 M2 管脚添加内部上拉、下拉或者两者都不添加。选择 Pullnone 以禁用 M2 管脚上的上拉电阻器和下拉电阻器。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	无	<字符串>	此设置用于在 MultiBoot 设置（存储在 WBSTAR 寄存器中）中设置下一次配置的起始地址。

表 42：**Zynq 7000 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable 和 Disable	设置为 Disable 时，将从 `.bit` 文件中移除 IPROG 命令。这样即可允许在上电时加载黄金镜像，而不是跳转到多重启动配置中的多重启动镜像。
BITSTREAM.CONFIG.OVERTEMPPOWERDOWN	Disable	Disable 和 Enable	此设置用于在 XADC 检测到温度超出可接受的最大工作温度时关闭器件。需为 XADC 设置外部电路才能使用该选项。
BITSTREAM.CONFIG.PERSIST	No	No 和 Yes	此设置用于在完成配置后，保留对多功能配置管脚的配置逻辑访问权。此设置主要用于在配置后保留 SelectMAP 端口用于回读访问，但也可配合任意配置模式使用。对于 JTAG 配置则无需保留 (PERSIST)，因为 JTAG 端口是专用端口且始终可用。PERSIST 和 ICAP 不能同时使用。 请参阅用户指南以获取相关说明。针对使用 SelectMAP 配置管脚的“Readback”（回读）和“Partial Reconfiguration”（部分重配置）操作，需使用 PERSIST，并且使用 SelectMAP 模式或串行模式时，也应使用 PERSIST。
BITSTREAM.CONFIG.REVISIONSELECT	00	00、01、10 或 11	此设置用于在温态启动起始地址 (WBSTAR) 寄存器中为下一次温态启动指定 RS[1:0] 设置的内部值。
BITSTREAM.CONFIG.REVISIONSELECT_ TRISTATE	Disable	Disable 和 Enable	此设置用于通过在温态启动起始地址 (WBSTAR) 中设置相应选项来指定是否启用 RS[1:0] 三态。 RS[1:0] 管脚三态启用 0：启用 RS 三态 1：禁用 RS 三态
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable 和 Disable	此设置用于启用或禁用 SelectMAP 模式“Abort”（异常中止）序列。如果禁用，则忽略器件管脚上的 Abort 序列。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No 和 Yes	此设置启用 SPI 32 位地址样式，对于存储空间不小于 256 Mb 的 SPI 器件，此设置是必需的。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2 和 4	针对来自第三方 SPI 闪存器件的“Master SPI”（主 SPI）配置，将 SPI 总线设置为“Dual (x2)”（双通道）或“Quad (x4)”（四通道）模式。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No 和 Yes	此设置用于将 FPGA 设置为使用下降沿时钟进行 SPI 数据捕获。这样可改进时序裕度，并能够提升配置的时钟速率。
BITSTREAM.CONFIG.TCKPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TCK 管脚添加上拉、下拉或两者都不添加，此管脚是 JTAG 测试时钟。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDIPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDI 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和 JTAG 寄存器的串行数据输入。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TDOPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TDO 管脚添加上拉、下拉或两者都不添加，此管脚是所有 JTAG 指令和数据寄存器的串行数据输出。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.TIMER_CFG	无	<8 个数字的十六进制字符串>	在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_USR 同时使用。
BITSTREAM.CONFIG.TIMER_USR	0x00000000	<8 个数字的十六进制字符串>	在配置模式下启用看门狗定时器，并设置值。该选项不能与 TIMER_CFG 同时使用。
BITSTREAM.CONFIG.TMSPIN[1]	Pullup	Pullup、Pulldown 和 Pullnone	此设置用于向 TMS 管脚添加上拉、下拉或两者都不添加，此管脚是 TAP 控制器的模式输入信号。TAP 控制器可为 JTAG 提供控制逻辑。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。

表 42：**Zynq 7000 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown、Pullup 和 Pullnone	此设置用于向未使用的 SelectIO™ 管脚 (IOB) 添加上拉、下拉或者两者都不添加。它对于专用配置管脚无效。专用配置管脚列表因架构而异。Pullnone 设置显示与上拉和下拉之间都不存在任何连接。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	<8 个数字的十六进制字符串>	此设置用于识别实现的版本。您可在“User ID”（用户 ID）寄存器中添加最多 1 个含 8 个数字的十六进制字符串。
BITSTREAM.CONFIG.USR_ACCESS	无	<8 个数字的十六进制字符串> 和 TIMESTAMP	此设置用于将 1 个含 8 个数字的十六进制字符串或时间戳写入 AXSS 配置寄存器。时间戳值的格式为 ddddd MMMM yyyyyy hhhhh mmmmmm ssssss：对应日、月、年（2000 年 = 00000）、小时、分钟、秒。FPGA 互连结构可通过 USR_ACCESS 原语直接访问此寄存器的内容。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram 和 efuse	此设置用于判断将使用的 AES 加密密钥的位置：密钥来自于电池供电式 RAM (BBRAM) 或 eFUSE 寄存器（7 系列）。 仅当“Encrypt”选项设置为“True”时，该属性才可用。
BITSTREAM.GENERAL.COMPRESS	False	True 和 False	比特流中的多帧写入功能用于减小比特流的大小，而不是用于减小比特流 (`.bit`) 文件的大小。使用 COMPRESS 并不保证比特流大小会缩小。
BITSTREAM.GENERAL.CRC	Enable	Enable 和 Disable	此设置用于控制比特流中循环冗余校验 (CRC) 值的生成。设为启用后，可根据比特流内容计算出唯一的 CRC 值。如果计算所得的 CRC 值与比特流中的 CRC 值不匹配，那么器件将无法进行配置。禁用 CRC 时，将在比特流中插入常量值以代替 CRC，且器件不会计算 CRC。 CRC 默认值为“Enable”，除非 BITSTREAM.ENCRYPTION.ENCRYPT 设为“Yes”，在此情况下禁用 CRC。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No 和 Yes	此设置用于在完成配置后，禁用通过 JTAG 与 Boundary Scan (BSCAN) 块进行通信的功能。
BITSTREAM.GENERAL.JTAG_XADC	Enable	Enable、Disable 和 StatusOnly	此设置用于启用或禁用与 XADC 的 JTAG 连接。
BITSTREAM.GENERAL.XADCENHANCEDLINEARITY	Off	Off 和 On	此设置用于禁用部分内置数字校准功能，因为此类设置导致 INL 看上去比实际模拟性能更糟。
BITSTREAM.GENERAL.PERFRAMECRC	No	No 和 Yes	在比特流中按固定间隔插入 CRC 值。这些值用于指示传入比特流的完整性，并且可在将配置数据加载到器件中之前标记错误（如 ICAP 的 INIT_B 管脚和 PRERROR 端口上所示）。虽然对于部分比特流而言，该属性设置为 Yes 最为合适，但它可将 CRC 值插入所有比特流，包括完整器件比特流。
BITSTREAM.READBACK.ACTIVERECONFIG	No	No 和 Yes	此设置用于在配置期间，阻止断言 GHIGH 和 GSR 有效。这是动态部分重配置增强功能所必需的设置。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto、Top 和 Bottom	此设置用于选择顶部或底部 ICAP 端口。
BITSTREAM.READBACK.READBACK	False	True 和 False	此设置支持您通过创建必要的回读文件来执行回读功能。
BITSTREAM.READBACK.SECURITY	无	None、Level1 和 Level2	此设置用于指定是否禁用回读和重配置。 针对 Security 指定 Level1 即禁用回读。针对 Security 指定 Level2 即禁用回读和重配置。
BITSTREAM.READBACK.XADCPARTIALRECONFIG	Disable	Disable 和 Enable	禁用时，XADC 可在部分重配置期间持续工作。设为启用时，XADC 在部分重配置期间可在安全 (Safe) 模式下工作。
BITSTREAM.STARTUP.DONEPIPE	Yes	Yes 和 No	此设置用于告知 FPGA，等待 CFG_DONE (DONE) 管脚转至高电平，并等待首个时钟沿出现后，再转至 Done（完成）状态。

表 42：**Zynq 7000 比特流设置** (续)

设置	默认值	可能的值	描述
BITSTREAM.STARTUP.DONE_CYCLE	4	4、1、2、3、5、6 和 Keep	此设置用于选择激活 FPGA Done 信号的“Startup”（启动）阶段。当 DonePipe=Yes 时，则延迟转至“Done”（完成）状态。
BITSTREAM.STARTUP.GTS_CYCLE	5	5、1、2、3、4、6、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向 I/O 缓冲器释放内部三态控制。
BITSTREAM.STARTUP.GWE_CYCLE	6	6、1、2、3、4、5、Done 和 Keep	此设置用于选择 Startup 阶段，在所选阶段内将向触发器、LUT RAM 和移位寄存器断言内部写入使能有效。GWE_cycle 还会启用 BRAMS。在进入 Startup 阶段前，块 RAM 写入和读取都处于禁用状态。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait、0、1、2、3、4、5 和 6	此设置用于选择 Startup 阶段，在所选阶段内将等待至 DLL/DCM/PLL 锁定为止。如果选择 NoWait，则 Startup 顺序不会等待至 DLL/DCM/PLL 锁定。
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5 和 6	此设置用于指定在 Startup 周期内停滞，直至数控阻抗 (DCI) 匹配信号断言有效为止。DCI 匹配不会在 Match_cycle 上开始。Startup 序列会在此周期内等待至 DCI 匹配为止。鉴于判定 DCI 匹配所需时间过程中涉及多个变量，因此在任意给定系统中，完成 Startup 序列所需的 CCLK 周期数不尽相同。理想情况下，配置解决方案应持续驱动 CCLK 直至 DONE 转至“High”（高电平）为止。 当指定的设置为“Auto”（自动）时，`write_bitstream` 会搜索设计中是否包含任意 DCI I/O 标准。如果存在 DCI 标准，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=2。否则，`write_bitstream` 会使用 BITSTREAM.STARTUP.MATCH_CYCLE=NoWait。
BITSTREAM.STARTUP.STARTUPCLK	Cclk	Cclk、UserClk 和 JtagClk	器件配置后，StartupClk 序列可同步至 Cclk、用户时钟或 JTAG 时钟。默认值为 Cclk。 Cclk 允许您同步到 FPGA 器件中提供的内部时钟。 UserClk 允许您同步到用户定义的信号，此信号连接到 STARTUP 符号的 CLK 管脚。 JtagClk 允许您同步到 JTAG 提供的时钟。此时钟会对为 JTAG 提供控制逻辑的 TAP 控制器进行排序。

注释：

1. 对于专用配置管脚，AMD 建议您使用比特流默认设置。

Versal 自适应 SoC 可编程器件镜像 (PDI) 设置

下表所示 Versal 自适应 SoC 器件的器件配置设置可搭配 `set_property <Setting> <Value> [current_design]` Vivado 工具 Tcl 命令一起使用。

注释：在 Versal 自适应 SoC 架构上，原先支持将可编程器件镜像设置作为比特流设置来进行配置，现在其中大部分设置都改为在 Control, Interfaces, and Processing System (CIPS) IP 中进行配置，或者作为 Bootgen 设置来进行配置。如需了解更多信息，请参阅《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352) 或《Bootgen 用户指南》(UG1283)。例如，在 UltraScale 架构上，使用 `BITSTREAM.CONFIG.USERID` 比特流设置对 USERCODE 进行置位。要在 Versal 架构上对 USERCODE 进行置位，必须在 CIPS IP 上使用以下属性对其进行置位：`set_property CONFIG.PS_PMC_CONFIG {JTAG_USERCODE 0x<32 bit hex value>} [get_bd_cells / versal_cips_0]`

表 43：**Versal 自适应 SoC 可编程器件镜像设置**

设置	默认值	可能的值	描述
BITSTREAM.CONFIG.USR_ACCESS	None	None、<8 个数字的十六进制字符串 > 和 TIMESTAMP	此设置用于将 1 个含 8 个数字的十六进制字符串或时间戳写入 PLM_RTCA 模块中的 USR_ACCESS 寄存器。时间戳值的格式为 ddddd MMMM yyyyyy hhhhh mmmmmm ssssss：对应日、月、年（2000 年 = 00000）、小时、分钟、秒。可通过 PL 中的 PS 或 AXI Manager（AXI 管理器）直接访问此寄存器的内容。
BITSTREAM.GENERAL.COMPRESS	True	True 和 False	使用运行长度编码算法来减小 PL 配置数据的大小。大部分情况下，这可减小 PL 配置数据的大小。
BITSTREAM.GENERAL.CRC	False	True 和 False	此设置用于控制 PDI 的 PL 部分中循环冗余校验 (CRC) 值的生成。设为启用后，可根据 PDI 内容的 PL 部分来计算出唯一的 CRC 值。如果计算所得 CRC 值与 PDI 中的 CRC 值不匹配，那么器件将无法进行配置。
BITSTREAM.GENERAL.PERFRAMECRC	False	True 和 False	在 PDI 的 PL 部分中定期插入 CRC 值。这些值用于确认传入配置数据的完整性，并且可在将这些配置数据加载到器件中之前标记错误。该属性设置为 Yes 时可将 CRC 值插入所有 PDI（包括完整器件镜像），但此设置仅对部分 PDI 最为合适。

附录 B

触发器状态机语言描述

触发器状态机语言用于描述映射到 ILA 调试核的高级触发器逻辑的复杂触发条件。触发器状态机具有下列特性：

- 最多 16 种状态。
- 用于复杂状态转换的单向、双向和三向条件分支。
- 4 个内置 16 位计数器用于对事件进行计数、实现定时器等。
- 4 个内置标志 (flag)，用于监控触发器状态机执行状态。
- 触发器操作。

状态

每个状态机程序均可声明最多 16 种状态。每种状态均由状态声明和主体组成：

```
state <state_name>:     <state_body>
```

Goto 操作

`goto` 操作用于执行状态转换。以下是使用 `goto` 操作在触发前执行状态转换的示例：

```
state my_state_0:     goto my_state_1;   state my_state_1:     trigger;
```

条件分支

触发器状态机语言支持对应每个状态的单向、双向和三向条件分支。

- 单向分支涉及使用 `goto` 操作，且其中不含任何 `if/elseif/else/endif` 构造：

  ```
  state my_state_0:       goto my_state_1;
  ```

- 双向条件分支使用 `goto` 操作，且其中包含 `if/else/endif` 构造：

  ```
  state my_state_0:     if (<condition1>) then       goto my_state_1;
      else       goto my_state_0;     endif
  ```

- 三向条件分支使用 `goto` 操作，且其中包含 `if/else/elseif/endif` 构造：

```
  state my_state_0:     if (<condition1>) then       goto my_state_1;
    elseif (<condition2>) then       goto my_state_2;     else       goto
my_state_0;     endif
```

如需了解有关如何使用 `<condition1>` 和 `<condition2>` 来构建先前所示条件语句的更多信息，请参阅“条件语句”部分。

相关信息

条件语句

计数器

4 个内置 16 位计数器都具有固定名称，分别为 `$counter0`、`$counter1`、`$counter2` 和 `$counter3`。计数器可复位、递增，也可用于条件语句。

- 要将计数器复位，请使用 `reset_counter` 操作：

```
state my_state_0:   reset_counter $counter0;       goto my_state_1;
```

- 要使计数器递增，请使用 `increment_counter` 操作：

```
state my_state_0:   increment_counter $counter3;       goto my_state_1;
```

如需了解有关如何在条件语句中使用计数器的更多信息，请参阅 条件语句。

相关信息

条件语句

标志

标志 (Flag) 可用于监控触发器状态机程序执行时的进展情况。4 个内置标志都具有固定名称，分别为 `$flag0`、`$flag1`、`$flag2` 和 `$flag3`。标志可设置也可清除。

- 要设置标志，请使用 `set_flag` 操作：

```
state my_state_0:   set_flag $flag0;       goto my_state_1;
```

- 要清除标志，请使用 `clear_flag` 操作：

```
state my_state_0:   clear_flag $flag2;       goto my_state_1;
```

条件语句

调试探针条件

调试探针条件可在双向或三向分支条件语句中使用。每个调试探针条件都占用调试探针连接到的 ILA 的 PROBE 端口上的 1 个触发器比较器。

重要提示！每个 PROBE 端口都可包含 1 到 16 个触发器比较器（在编译时配置）。这意味着根据 PROBE 端口上配置的比较器数量，在整个触发器状态机程序中，您只能使用调试探针条件中的特定调试探针，且次数仅限 1 到 16 次。

调试探针条件由 1 个比较运算符和 1 个值组成。有效的调试探针条件比较运算符包括：

- ==（等于）
- !=（不等于）
- >（大于）
- <（小于）
- >=（大于或等于）
- <=（小于或等于）

有效值格式如下：

```
<bit_width>'<radix><value>
```

其中：

- `<bit width>` 表示探针的宽度（以位数为单位）
- `<radix>` 为以下值之一
  - `b`（二进制）
  - `h`（十六进制）
  - `u`（无符号十进制）
- `<value>` 为以下值之一
  - `0`（逻辑 0）
  - `1`（逻辑 1）
  - `X`（忽略）
  - `R`（0 到 1 转换）- 仅对 1 位探针有效
  - `F`（1 到 0 转换）- 仅对 1 位探针有效
  - `B`（双向转换）- 仅对 1 位探针有效
  - `N`（无转换）- 仅对 1 位探针有效

  有效的调试探针条件值示例如下：

- 1 位二进制值 0

```
1'b0
```

- 12 位十六进制值 7A

```
12'h07A
```

- 9 位整数值 123

```
9'u123
```

调试探针条件语句示例如下：

- 单一位调试探针 `abc` 等于 0

```
if (abc == 1'b0) then
```

- 23 位调试探针 `xyz` 等于 456

```
if (xyz >= 23'u456) then
```

- 23 位调试探针 `klm` 不等于十六进制值 A5

```
if (klm != 23'h0000A5) then
```

多重调试探针条件语句示例如下：

- 2 个调试探针比较，使用“OR”函数组合：

```
if ((xyz >= 23'u456) || (abc == 1'b0)) then
```

- 2 个调试探针比较，使用“AND”函数组合：

```
if ((xyz >= 23'u456) && (abc == 1'b0)) then
```

- 3 个调试探针比较，使用“OR”函数组合：

```
if ((xyz >= 23'u456) || (abc == 1'b0) || (klm != 23'h0000A5)) then
```

- 3 个调试探针比较，使用“AND”函数组合：

```
if ((xyz >= 23'u456) && (abc == 1'b0) && (klm != 23'h0000A5)) then
```

计数器条件

计数器条件可在双向或三向分支条件语句中使用。每个计数器条件都占用 1 个计数器比较器。

重要提示！每个计数器仅含 1 个计数器比较器。这意味着，在整个触发器状态机程序中，每个特定计数器在任一计数器条件中只能使用一次。

探针端口条件由 1 个比较运算符和 1 个值组成。有效的探针条件比较运算符包括：

- ==（等于）
- !=（不等于）

重要提示！每个计数器位宽始终为 16 位。

有效的计数器条件值示例如下：

- 16 位二进制值 0

```
16'b0000_0000_0000_0000 16'b0000000000000000
```

- 16 位十六进制值 7A

```
16'h007A
```

- 16 位整数值 123

```
16'u123
```

计数器条件语句示例：

- 计数器 `$counter0` 等于二进制 0

```
($counter0 == 16'b0000000000000000)
```

- 计数器 `$counter2` 不等于十进制 23

```
($counter2 != 16'u23)
```

调试探针条件与计数器条件的组合

调试探针条件与计数器条件可使用如下规则组合在一起以形成单一条件：

- 所有调试探针比较都必须使用相同的“||”(OR) 或“&&”(AND) 运算符组合在一起。
- 组合后的调试探针可使用“||”(OR) 或“&&”(AND) 运算符与计数器条件加以组合，与用于组合调试探针比较操作的运算符无关。

包含多个调试探针和计数器条件的语句示例如下：

- 2 个调试探针比较运算使用“OR”函数组合，然后使用“AND”函数与计数器条件组合：

```
if (((xyz >= 23'u456) || (abc == 1'b0)) && ($counter0 == 16'u0023)) then
```

- 2 个调试探针比较运算使用“AND”函数组合，然后使用“OR”函数与计数器条件组合：

```
if (((xyz >= 23'u456) && (abc == 1'b0)) || ($counter0 == 16'u0023)) then
```

- 3 个调试探针比较运算使用“OR”函数组合，然后使用“AND”函数与计数器条件组合：

```
if (((xyz >= 23'u456) || (abc == 1'b0) || (klm != 23'h0000A5)) &&
($counter0 == 16'u0023)) then
```

- 3 个调试探针比较运算使用“AND”函数组合，然后使用“OR”函数与计数器条件组合：

```
if (((xyz >= 23'u456) && (abc == 1'b0) && (klm != 23'h0000A5)) ||
($counter0 == 16'u0023)) then
```

触发器状态机语言语法

注释：

- 该语言区分大小写

- 注释字符为井号“#”字符。包含 # 字符以及该字符后的所有内容都将被忽略。
- 'THING' = THING 表示终端
- {<thing>} = 0 或更多 thing
- [<thing>] = 0 或 1 个 thing

```
<program> ::= <state_list> <state_list> ::= <state_list> <state> |
<state> <state> ::= 'STATE' <state_label> ':' <if_condition> |
<action_block>
```

```
<action_block> ::= <action_list> 'GOTO' <state_label> ';' | <action_list>
'TRIGGER' ';' | 'GOTO' <state_label> ';' | 'TRIGGER' ';' <action_list> ::=
<action_statement> | <action_list> <action_statement>
<action_statement> ::= 'SET_FLAG' <flag_name> ';' | 'CLEAR_FLAG'
<flag_name> ';' | 'INCREMENT_COUNTER' <counter_name> ';' | 'RESET_COUNTER'
<counter_name> ';' <if_condition> ::= 'IF' '(' <condition> ')' 'THEN'
<actionblock>                    ['ELSEIF' '(' <condition> ')' 'THEN'
<actionblock>]                    'ELSE' <actionblock>
                  'ENDIF' <condition> ::= <probe_match_list> |
<counter_match> | <probe_counter_match> <probe_counter_match> ::= '('
<probe_counter_match> ')' | <probe_match_list> <boolean_logic_op>
<counter_match> | <counter_match> <boolean_logic_op> <probe_match_list>
<probe_match_list> ::= '(' <probe_match> ')' | <probe_match>
<probe_match> ::= <probe_match_list> <boolean_logic_op> <probe_match_list>
| <probe_name> <compare_op> <constant> | <constant> <compare_op>
<probe_name> <counter_match> ::= '(' <counter_match> ')' | <counter_name>
<compare_op> <constant> | <constant> <compare_op> <counter_name>
<constant> ::= <integer_constant> | <hex_constant> | <binary_constant>
<compare_op> ::= '==' | '!=' | '>' | '>=' | '<' | '<='
<boolean_logic_op> ::= '&&' | '||' --- The following are in regular
expression format to simplify expressions: --- [A-Z0-9] means match any
single character in AB...Z,0..9 --- [AB]+ means match [AB] one or more
times like A, AB, ABAB, AAA, etc --- [AB]* means match [AB] zero or more
times <probe_name> ::= [A-Z_\[\]<>/][A-Z_0-9\[\]<>/]+ <state_label> ::= [A-
Z_][A-Z_0-9]+ <flag_name> ::= \$FLAG[0-3] <counter_name> ::= \$COUNTER[0-3]
<hex_constant> ::= <integer>*'h<hex_digit>+ <binary_constant> ::=
<integer>*'b<binary_digit>+ <integer_constant> ::=
<integer>*'u<integer_digit>+ <integer> ::= <digit>+ <hex_digit> ::=
[0-9ABCDEFBN_] <binary_digit> ::= [01XRFBN_] <digit> ::= [0-9]
```

低级别 SVF JTAG 命令

注释：在 AMD Versal™ 器件上不支持 SVF。

低级别 JTAG 命令允许您扫描多个 FPGA JTAG 链。针对链操作所生成的 SVF 命令使用这些低级别命令来访问链中的 FPGA。

本附录提供了这些命令的概述。在位于以下网址的《Serial Vector Format Specification》文档内提供了详细解释：http://www.jtagtest.com/pdf/svf_specification.pdf。

报头数据寄存器 (HDR) 和报头指令寄存器 (HIR)

语法

```
HDR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)]; HIR
length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];
```

用途

指定报头模式，默认在每次扫描操作前移入此模式。报头模式用于指定如何使用一组前导位来填充扫描语句，以供容纳位于扫描路径上超出感兴趣组件范围的器件。

常规信息

“Header Data Register (HDR)”（报头数据寄存器）用于指定要追加到所有后续 SDR 命令开头之前的默认报头模式。“Header Instruction Register (HIR)”（报头指令寄存器）用于指定要添加到所有后续 SIR 命令开头的默认报头模式。报头命令具有一组与之相对的报尾命令（TIR 和 TDR），下一章节中描述了这些报尾命令。可通过将报头长度设置为 0 来移除报头。

报尾数据寄存器 (TDR) 和报尾指令寄存器 (TIR)

语法

```
TDR length [TDI (tdi)] [TDO (tdo)][MASK (mask)] [SMASK (smask)]; TIR length
[TDI (tdi)] [TDO (tdo)][MASK (mask)] [SMASK (smask)];
```

用途

指定报尾模式，默认在所有后续扫描操作完成后移入此模式。报尾模式用于指定如何使用一组尾位元来填充扫描语句，以供容纳位于扫描路径上的感兴趣组件之后的器件。

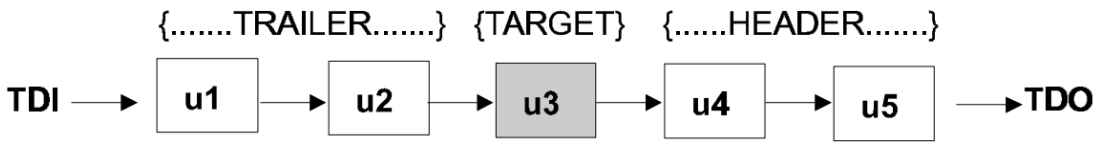
常规信息

“Trailer Data Register (TDR)”（报尾数据寄存器）用于指定要追加到所有后续 SDR 命令末尾的报尾模式。“Trailer Instruction Register (TIR)”（报尾指令寄存器）用于指定将追加到所有后续 SIR 命令末尾的默认报尾模式。可通过将报尾长度设置为 0 来移除报尾。

示例

在此示例中，专为 ASIC 开发了 1 个 SVF 文件。将此 ASIC 布局在开发板上作为 u3，如下所示：

图 187：**TDR 示例**



如果相应的头尾语句定义为适用于 u3 前后的器件，那么只需对原先为 ASIC 开发的一组 SVF 语句进行少量修改即可复用。在此示例中，将为器件 u4 和 u5 定义报头模式，并为 u2 和 u1 定义报尾模式。可选参数可按任意顺序指定。每个可选参数都仅限指定一次。针对 TDI、TDO、MASK 或 SMASK 指定的十六进制字符串值不得大于长度参数所暗示的最大值。如果未明确指定，那么假定十六进制字符串包含前导零位。

scan_ir_hw

在 `hw_jtag` 上执行移位 IR。

语法

```
scan_ir_hw_jtag [-tdi <arg>] [-tdo <arg>] [-mask <arg>] [-smask <arg>] [-
quiet] [-verbose] <length>
```

常规信息

`scan_ir_hw_jtag` 命令用于指定将扫描到 JTAG 接口目标指令寄存器中的扫描模式。此命令以 `hw_jtag` 对象为目标，该对象是在 JTAG 模式下使用 `open_hw_target -jtag_mode` 命令打开 `hw_target` 时创建的。切换至 scan_ir_hw_jtag 命令中指定的扫描模式之前，以 `hw_jtag` 对象为目标时，最后定义的报头属性 (HIR) 将追加到指定的数据模式开头位置之前。最后定义的报尾属性 (TIR) 则追加到数据模式的末尾之后。

针对 `-tdi`、`-tdo`、`-mask` 或 `-smask` 指定的十六进制字符串所表示的位数不能大于 `<length>` 所指定的最大值。

scan_ir_hw_jtag 命令用于返回十六进制阵列，其中包含从 hw_jtag 捕获的 TDO 数据，或者如果捕获失败，则返回错误。

示例

以下示例扫描 JTAG 指令寄存器中的 24 位值：

```
scan_ir_hw_jtag 24
```

以下示例先向 TDI 发送 24 位值 0x00_0010（LSB 优先），然后捕获 TDO 输出，以 0xF3_FFFF 应用掩码，并将返回的 TDO 值与指定值 tdo 0x81_8181 进行比较。

```
scan_ir_hw_jtag 24 -tdi 000010 -tdo 818181 -mask F3FFFF -smask 0
```

scan_dr_hw

在 hw_jtag 上执行移位 DR。

语法

```
scan_dr_hw_jtag [-tdi <arg>] [-tdo <arg>] [-mask <arg>] [-smask <arg>] [-
quiet] [-verbose] <length>
```

常规信息

scan_dr_hw_jtag 命令用于指定将扫描到 JTAG 接口目标数据寄存器中的扫描模式。此命令以 hw_jtag 对象为目标，该对象是在 JTAG 模式下使用 open_hw_target -jtag_mode 命令打开 hw_target 时创建的。切换 scan_dr_hw_jtag 命令中指定的扫描模式之前，以 hw_jtag 对象为目标时，最后定义的报头属性 (HDR) 将追加到指定的数据模式开头位置之前。最后定义的报尾属性 (TDR) 则追加到数据模式的末尾之后。

scan_dr_hw_jtag 命令用于返回十六进制阵列，其中包含从 hw_jtag 捕获的 TDO 数据，或者如果捕获失败，则返回错误。

示例

以下示例扫描 JTAG 数据寄存器中的 24 位值：

```
    scan_dr_hw_jtag 24
```

以下示例先向 TDI 发送 24 位值 0x00_0010（LSB 优先），然后捕获数据输出 TDO，以 0xF3_FFFF 应用掩码，并将返回的 TDO 值与指定值 -tdo 0x81_8181 进行比较。

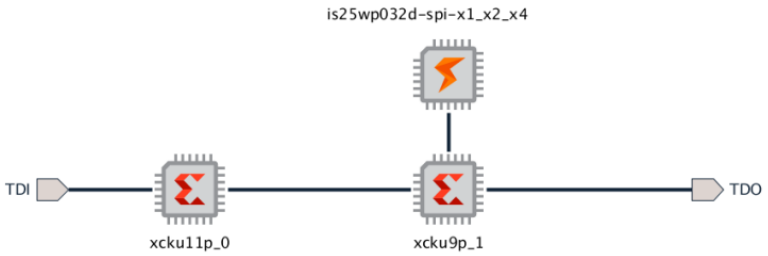
```
   scan_dr_hw_jtag 24 -tdi 000010 -tdo 818181 -mask F3FFFF -smask 0
```

多链 SVF 操作

以下示例显示了如何在 SVF 链上处理操作。

每个链中连接有 2 个器件：xcku11 和 xcku9。配置存储器连接到链中的第二个器件 (xcku9)。为访问此配置存储器，SVF 会使用 HIR、HDR、TIR 和 TDR 命令来生成命令。为烧录此配置存储器所生成的命令会考量链长度，并将此信息整合到低级别 JTAG 操作中。

图 188：**多链 SVF 操作示例**



生成的 `.svf` 文件包含以下操作：

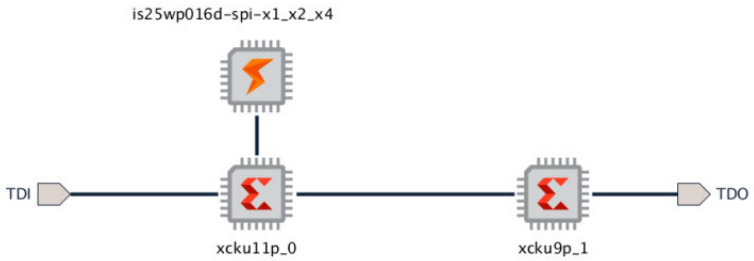
```
HIR 0 ; TIR 6 TDI (3f) SMASK (3f) ; HDR 0 ; TDR 1 TDI (00) SMASK (01) ; //
config/idcode SIR 6 TDI (09) ; SDR 32 TDI (00000000) TDO (0484a093) MASK
(0fffffff) ; // config/jprog STATE RESET; STATE IDLE; SIR 6 TDI (0b) ; SIR
6 TDI (14) ; // Modify the below delay for config_init operation (0.100000
sec typical, 0.100000 sec maximum) RUNTEST 0.100000 SEC; // config/jprog/
poll RUNTEST 10000 TCK; SIR 6 TDI (14) TDO (11) MASK (31) ; // config/slr
SIR 6 TDI (05) ;
```

配置存储器连接至链中第一个器件的多链 SVF 操作

在此示例中，为接入 ku9 器件，对 TIR 和 TDR 指令所用的 SMASK 值为 0000 0011 1111 (0x3f)。为接入链中的第二个器件，首先推送掩码值，然后推送 SIR 和 SDR 指令。SIR 和 SDR 指令会将 HIR、HDR、TIR 和 TDR 信息相组合。

如果连接到第一个器件 (xcku11) 的配置存储器需烧录，那么 SVF 生成的命令会如下发生改变：

图 189：**配置存储器连接至链中第一个器件的多链 SVF 操作示例**



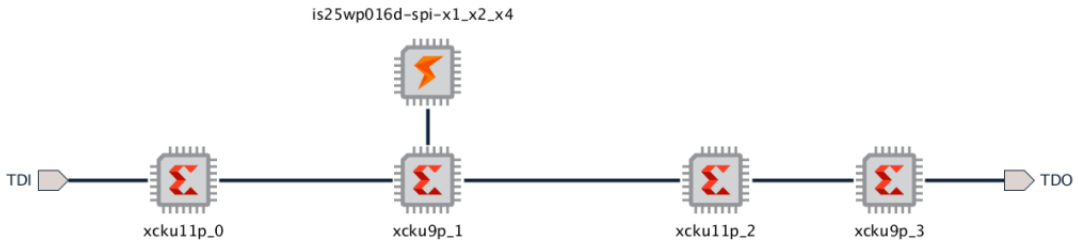

```
HIR 6 TDI (3f) SMASK (3f) ; TIR 0 ; HDR 1 TDI (00) SMASK (01) ; TDR 0 ; //
config/idcode SIR 6 TDI (09) ; SDR 32 TDI (00000000) TDO (04a4e093) MASK
(0fffffff) ; // config/jprog STATE RESET; STATE IDLE; SIR 6 TDI (0b) ; SIR
6 TDI (14) ; // Modify the below delay for config_init operation (0.100000
sec typical, 0.100000 sec maximum) RUNTEST 0.100000 SEC; // config/jprog/
poll RUNTEST 10000 TCK; SIR 6 TDI (14) TDO (11) MASK (31) ; // config/slr
SIR 6 TDI (05) ;
```

配置存储器连接至链中第二个器件的多链 SVF 操作

在此示例中，为接入 ku11 器件，对 HIR 和 HDR 指令所用的 SMASK 值为 0011 1111 (0x3f)。为访问链中的首个器件，首先推送掩码值，然后推送 SIR 和 SDR 指令。SIR 和 SDR 指令会将 HIR、HDR、TIR 和 TDR 信息相组合。

现在，假设链中已连接 4 个器件：xcku11、xcku9、xcku11 和 xcku9。配置存储器连接至此链中的第二个器件 (xcku9)，在此情况下要接入该器件，需同时使用如下 HIR 和 TIR 指令：

图 190：**多链 SVF 操作示例 - 链中的第二个器件**

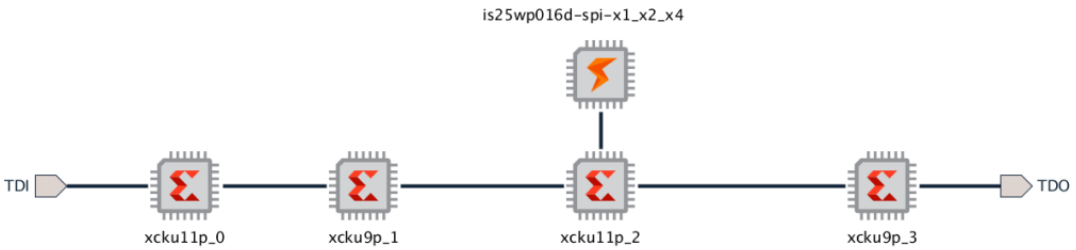



```
HIR 12 TDI (0fff) SMASK (0fff) ; TIR 6 TDI (3f) SMASK (3f) ; HDR 2 TDI (00)
SMASK (03) ; TDR 1 TDI (00) SMASK (01) ; // config/idcode SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (0484a093) MASK (0fffffff) ; // config/jprog
STATE RESET; STATE IDLE; SIR 6 TDI (0b) ; SIR 6 TDI (14) ; // Modify the
below delay for config_init operation (0.100000 sec typical, 0.100000 sec
maximum) RUNTEST 0.100000 SEC; // config/jprog/poll RUNTEST 10000 TCK; SIR
6 TDI (14) TDO (11) MASK (31) ; // config/slr SIR 6 TDI (05) ;
```

配置存储器连接至链中第三个器件的多链 SVF 操作

如果配置存储器连接到链中的第三个器件 (xcku9)，则执行以下操作。

图 191：**多链 SVF 操作示例 - 链中的第三个器件**




```
HIR 6 TDI (3f) SMASK (3f) ; TIR 12 TDI (0fff) SMASK (0fff) ; HDR 1 TDI (00)
SMASK (01) ; TDR 2 TDI (00) SMASK (03) ; // config/idcode SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (04a4e093) MASK (0fffffff) ; // config/jprog
STATE RESET; STATE IDLE; SIR 6 TDI (0b) ; SIR 6 TDI (14) ; // Modify the
below delay for config_init operation (0.100000 sec typical, 0.100000 sec
maximum) RUNTEST 0.100000 SEC; // config/jprog/poll RUNTEST 10000 TCK; SIR
6 TDI (14) TDO (11) MASK (31) ; // config/slr SIR 6 TDI (05) ;
```

hw_server 支持的 JTAG 线缆和器件

hw_server 支持的兼容 JTAG 下载线缆和器件列表如下：

- AMD SmartLynq+ 模块 (HW-SMARTLYNQ-PLUS-G)
- AMD SmartLynq 数据线缆 (HW-SMARTLYNQ-G/DLC20)
- AMD 平台电缆 USB II (DLC10)
- AMD 平台电缆 USB（DLC9G、DLC9LP 和 DLC9）
- Digilent JTAG-HS1
- Digilent JTAG-HS2
- Digilent JTAG-HS3
- Digilent JTAG-SMT1
- Digilent JTAG-SMT2

用于 Vivado 硬件管理器支持的 FTDI 器件烧录

对于在 AMD JTAG 软件工具（如 XSDB 或 AMD Vivado™ Hardware Manger）中识别为 USB-to-JTAG 接口的 FTDI 器件，必须烧写 FTDI 器件上的 EEPROM。要进行 FTDI 烧录，请使用 Vivado 安装中包含的 `program_ftdi` 实用工具来完成该操作。完成烧录后，在 Vivado 中会将此 FTDI 器件识别为有效的烧录电缆。

注释：如需了解板上实现的详细信息（包括 FTDI 连接方式），请参阅 XTP610 中提供的 AMD VCK190 板级原理图：https://china.xilinx.com/products/boards-and-kits/vck190.html。请确保 FTDI 连接（包括 ADBUS0-7）与 VCK190 实现相匹配。

注释：`program_ftdi` 实用工具将对要用作为 USB-to-JTAG 接口的 Channel A（通道 A）进行配置。如果器件具有多个通道（以 FT4232H 为例），那么这些通道会配置为默认 RS232 UART 模式。如果不希望使用默认 RS232 UART 模式，那么可以使用 FTDI 提供的 FT_PROG EEPROM 实用工具来为任意未使用的通道更改端口配置，这样不会影响 AMD JTAG 软件工具中识别 FTDI 器件的能力。

`program_ftdi` 实用工具支持以下 FTDI 工具：

- FT232H
- FT2232H
- FT4232H

命令参考如下所示。

```
****** program_ftdi v2023.1 **** Build date : Apr 16 2023-14:45:22 **
Copyright 1986-2022 Xilinx, Inc. All Rights Reserved. ** Copyright
2022-2023 Advanced Micro Devices, Inc. All Rights Reserved. Short
Description: Write/Read to FTDI EEPROM for Xilinx JTAG Tools support
Syntax: program_ftdi {-write -ftdi=<ftdi_part> -serial=<serial_number>
[options] | -write -filein=<cfg_filein> | -read [-fileout=<cfg_fileout>] | -
erase} [-help] options: Name Description
------------------------------------------------------------------------------
-------------------------- -f, -fdti Specify the ftdi device to be
programmed <FT232H | FT2232H | FT4232H> -s, -serial Serial number to be
written into the EEPROM [-v, --vendor] Vendor information [-b, --board]
Name of the board/product being programmed [-d, -desc, -description] A
short description of the board -fi, -filein Input file with all fields to
be written [-fo, -fileout] File to which the FDI EEPROM should be read back
[-lh, -longhelp] Get long help description for program_ftdi util Examples:
program_ftdi -write -ftdi FT2232H -serial 0ABC01 -vendor "my vendor co" -
board "my board" -desc "my product desc" program_ftdi -write -filein
<path_to_config_file> program_ftdi -read program_ftdi -read -fileout
<path_to_config_file> program_ftdi -erase
```

注释：请务必在烧录前验证并更新先前示例中的“Vendor”（供应商）、“Board”（开发板）、“Description”（描述）和“Serial Number”（序列号）字段。

附录 F

配置存储器支持

本章主要讲解了 AMD Vivado™ 软件支持的各种非易失性器件存储器。请使用本章作为指南，按 AMD 系列、接口、制造商、密度和数据宽度来为您的应用选择适用的配置存储器器件。

Artix 7 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 AMD Artix™ 7 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 44：**支持用于 Artix 7 BPI 的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	227e	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	227e	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxp	s70gl02gp	1	227e	2248	2201	2048	x16
Infineon	s29glxxxs	s29gl01gs	1	227e	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	227e	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	227e	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	227e	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	227e	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	227e	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	227e	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	227e	2222	2201	256	x16 和 x8

表 44：**支持用于 Artix 7 BPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	g18	28f128g18f	89	8900	NULL	NULL	128	x16
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	88b0	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	8901	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	887e	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	227e	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	227e	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	227e	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	227e	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	227e	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	227e	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	227e	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	227e	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	227e	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w640gh	20	227e	220c	2201	64	x16 和 x8
Micron	m29w	m29w640gl	20	227e	220c	2200	64	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	227e	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	227e	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	227e	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	227e	2223	2201	512	x16 和 x8

表 44：**支持用于 Artix 7 BPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	mt28fw	mt28fw02gb	89	227e	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	8963	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	899a	NULL	NULL	1024	x16
Micron	p30	28f00ap30t	89	8962	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	899a	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	881b	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	8818	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	891c	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	8919	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	8961	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	8999	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	8960	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	881a	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	8817	NULL	NULL	64	x16
Micron	p33	28f00ap33b	89	8967	NULL	NULL	1024	x16
Micron	p33	28f00ap33e	89	899f	NULL	NULL	1024	x16
Micron	p33	28f00ap33t	89	8966	NULL	NULL	1024	x16
Micron	p33	28f128p33b	89	8821	NULL	NULL	128	x16
Micron	p33	28f128p33t	89	881e	NULL	NULL	128	x16
Micron	p33	28f256p33b	89	8922	NULL	NULL	256	x16
Micron	p33	28f256p33t	89	891f	NULL	NULL	256	x16
Micron	p33	28f512p33b	89	8965	NULL	NULL	512	x16
Micron	p33	28f512p33e	89	899e	NULL	NULL	512	x16
Micron	p33	28f512p33t	89	8964	NULL	NULL	512	x16
Micron	p33	28f640p33b	89	8820	NULL	NULL	64	x16
Micron	p33	28f640p33t	89	881d	NULL	NULL	64	x16

表 45：**支持用于 Artix 7 SPI 的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2 和 x4
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2 和 x4
ISSI	is25lp	is25lp016d	9d	60	15	16	x1、x2 和 x4
ISSI	is25lp	is25lp032d	9d	60	16	32	x1、x2 和 x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1、x2 和 x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1、x2 和 x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1、x2 和 x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2 和 x4
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2 和 x4
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2 和 x4
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2 和 x4
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2 和 x4
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4

表 45：**支持用于 Artix 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2 和 x4
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2 和 x4
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2 和 x4
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2 和 x4
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2 和 x4
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4]	c2	20	18	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4]	c2	20	19	256	x1、x2 和 x4
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4]	c2	20	16	32	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4]	c2	20	1a	512	x1、x2 和 x4
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4]	c2	20	17	64	x1、x2 和 x4

表 45：**支持用于 Artix 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25v1635f	c2	23	15	16	x1、x2 和 x4
Macronix	mx25l	mx25v8035f	c2	23	14	8	x1、x2 和 x4
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4]	c2	25	38	128	x1、x2 和 x4
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4]	c2	25	35	16	x1、x2 和 x4
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4]	c2	25	39	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4]	c2	25	36	32	x1、x2 和 x4
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4]	c2	25	3a	512	x1、x2 和 x4
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4]	c2	25	37	64	x1、x2 和 x4
Macronix	mx25u	mx25u8035f [mx25u8033e-spi-x1_x2_x4]	c2	25	34	8	x1、x2 和 x4
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1、x2 和 x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1、x2 和 x4
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2 和 x4
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2 和 x4
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1、x2 和 x4
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1、x2 和 x4

表 45：**支持用于 Artix 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	20	ba	18	128	x1、x2 和 x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	20	ba	19	256	x1、x2 和 x4
Micron	mt25ql	mt25ql512	20	ba	20	512	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	20	bb	18	128	x1、x2 和 x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	20	bb	19	256	x1、x2 和 x4
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4

Kintex 7 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 AMD Kintex™ 7 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 46：**支持用于 Kintex 7 BPI 的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	227e	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	227e	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxp	s70gl02gp	1	227e	2248	2201	2048	x16
Infineon	s29glxxxs	s29gl01gs	1	227e	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	227e	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	227e	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	227e	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	227e	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	227e	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	227e	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	227e	2222	2201	256	x16 和 x8

表 46：**支持用于 Kintex 7 BPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	g18	28f128g18f	89	8900	NULL	NULL	128	x16
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	88b0	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	8901	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	887e	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	227e	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	227e	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	227e	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	227e	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	227e	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	227e	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	227e	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	227e	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	227e	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w640gh	20	227e	220c	2201	64	x16 和 x8
Micron	m29w	m29w640gl	20	227e	220c	2200	64	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	227e	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	227e	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	227e	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	227e	2223	2201	512	x16 和 x8

表 46：**支持用于 Kintex 7 BPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	mt28fw	mt28fw02gb	89	227e	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	8963	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	899a	NULL	NULL	1024	x16
Micron	p30	28f00ap30t	89	8962	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	899a	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	881b	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	8818	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	891c	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	8919	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	8961	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	8999	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	8960	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	881a	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	8817	NULL	NULL	64	x16
Micron	p33	28f00ap33b	89	8967	NULL	NULL	1024	x16
Micron	p33	28f00ap33e	89	899f	NULL	NULL	1024	x16
Micron	p33	28f00ap33t	89	8966	NULL	NULL	1024	x16
Micron	p33	28f128p33b	89	8821	NULL	NULL	128	x16
Micron	p33	28f128p33t	89	881e	NULL	NULL	128	x16
Micron	p33	28f256p33b	89	8922	NULL	NULL	256	x16
Micron	p33	28f256p33t	89	891f	NULL	NULL	256	x16
Micron	p33	28f512p33b	89	8965	NULL	NULL	512	x16
Micron	p33	28f512p33e	89	899e	NULL	NULL	512	x16
Micron	p33	28f512p33t	89	8964	NULL	NULL	512	x16
Micron	p33	28f640p33b	89	8820	NULL	NULL	64	x16
Micron	p33	28f640p33t	89	881d	NULL	NULL	64	x16

表 47：**支持用于 Kintex 7 SPI 的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2 和 x4
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2 和 x4
ISSI	is25lp	is25lp016d	9d	60	15	16	x1、x2 和 x4
ISSI	is25lp	is25lp032d	9d	60	16	32	x1、x2 和 x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1、x2 和 x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1、x2 和 x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1、x2 和 x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2 和 x4
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2 和 x4
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2 和 x4
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2 和 x4
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2 和 x4
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4

表 47：**支持用于 Kintex 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2 和 x4
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2 和 x4
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2 和 x4
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2 和 x4
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2 和 x4
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4]	c2	20	18	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4]	c2	20	19	256	x1、x2 和 x4
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4]	c2	20	16	32	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4]	c2	20	1a	512	x1、x2 和 x4
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4]	c2	20	17	64	x1、x2 和 x4

表 47：**支持用于 Kintex 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25v1635f	c2	23	15	16	x1、x2 和 x4
Macronix	mx25l	mx25v8035f	c2	23	14	8	x1、x2 和 x4
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4]	c2	25	38	128	x1、x2 和 x4
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4]	c2	25	35	16	x1、x2 和 x4
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4]	c2	25	39	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4]	c2	25	36	32	x1、x2 和 x4
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4]	c2	25	3a	512	x1、x2 和 x4
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4]	c2	25	37	64	x1、x2 和 x4
Macronix	mx25u	mx25u8035f [mx25u8033e-spi-x1_x2_x4]	c2	25	34	8	x1、x2 和 x4
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1、x2 和 x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1、x2 和 x4
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2 和 x4
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2 和 x4
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1、x2 和 x4
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1、x2 和 x4

表 47：**支持用于 Kintex 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	20	ba	18	128	x1、x2 和 x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	20	ba	19	256	x1、x2 和 x4
Micron	mt25ql	mt25ql512	20	ba	20	512	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	20	bb	18	128	x1、x2 和 x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	20	bb	19	256	x1、x2 和 x4
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4

Spartan 7 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 AMD Spartan™ 7 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

注释：串行外设接口 (SPI) 闪存是 Spartan 7 器件支持的配置存储器。Spartan 7 器件不支持字节宽度外设接口 (BPI) 闪存。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 48：**支持用于 Spartan 7 SPI 的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2 和 x4
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2 和 x4
ISSI	is25lp	is25lp016d	9d	60	15	16	x1、x2 和 x4
ISSI	is25lp	is25lp032d	9d	60	16	32	x1、x2 和 x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1、x2 和 x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1、x2 和 x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1、x2 和 x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2 和 x4
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2 和 x4
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2 和 x4
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4

表 48：**支持用于 Spartan 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2 和 x4
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2 和 x4
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2 和 x4
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2 和 x4
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2 和 x4
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2 和 x4
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2 和 x4

表 48：**支持用于 Spartan 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4]	c2	20	18	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4]	c2	20	19	256	x1、x2 和 x4
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4]	c2	20	16	32	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4]	c2	20	1a	512	x1、x2 和 x4
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4]	c2	20	17	64	x1、x2 和 x4
Macronix	mx25l	mx25v1635f	c2	23	15	16	x1、x2 和 x4
Macronix	mx25l	mx25v8035f	c2	23	14	8	x1、x2 和 x4
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4]	c2	25	38	128	x1、x2 和 x4
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4]	c2	25	35	16	x1、x2 和 x4
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4]	c2	25	39	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4]	c2	25	36	32	x1、x2 和 x4

表 48：**支持用于 Spartan 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4]	c2	25	3a	512	x1、x2 和 x4
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4]	c2	25	37	64	x1、x2 和 x4
Macronix	mx25u	mx25u8035f [mx25u8033e-spi-x1_x2_x4]	c2	25	34	8	x1、x2 和 x4
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1、x2 和 x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1、x2 和 x4
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2 和 x4
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2 和 x4
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1、x2 和 x4
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1、x2 和 x4
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	20	ba	18	128	x1、x2 和 x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	20	ba	19	256	x1、x2 和 x4
Micron	mt25ql	mt25ql512	20	ba	20	512	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	20	bb	18	128	x1、x2 和 x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	20	bb	19	256	x1、x2 和 x4
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4

表 48：**支持用于 Spartan 7 SPI 的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1、x2 和 x4

Virtex 7 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 AMD Virtex™ 7 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 49：**支持用于 Virtex 7 BPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	227e	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	227e	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxp	s70gl02gp	1	227e	2248	2201	2048	x16
Infineon	s29glxxxs	s29gl01gs	1	227e	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	227e	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	227e	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	227e	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	227e	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	227e	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	227e	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	227e	2222	2201	256	x16 和 x8

表 49：**支持用于 Virtex 7 BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	g18	28f128g18f	89	8900	NULL	NULL	128	x16
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	88b0	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	8901	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	887e	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	227e	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	227e	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	227e	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	227e	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	227e	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	227e	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	227e	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	227e	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	227e	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w640gh	20	227e	220c	2201	64	x16 和 x8
Micron	m29w	m29w640gl	20	227e	220c	2200	64	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	227e	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	227e	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	227e	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	227e	2223	2201	512	x16 和 x8

表 49：**支持用于 Virtex 7 BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	mt28fw	mt28fw02gb	89	227e	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	8963	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	899a	NULL	NULL	1024	x16
Micron	p30	28f00ap30t	89	8962	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	899a	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	881b	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	8818	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	891c	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	8919	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	8961	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	8999	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	8960	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	881a	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	8817	NULL	NULL	64	x16
Micron	p33	28f00ap33b	89	8967	NULL	NULL	1024	x16
Micron	p33	28f00ap33e	89	899f	NULL	NULL	1024	x16
Micron	p33	28f00ap33t	89	8966	NULL	NULL	1024	x16
Micron	p33	28f128p33b	89	8821	NULL	NULL	128	x16
Micron	p33	28f128p33t	89	881e	NULL	NULL	128	x16
Micron	p33	28f256p33b	89	8922	NULL	NULL	256	x16
Micron	p33	28f256p33t	89	891f	NULL	NULL	256	x16
Micron	p33	28f512p33b	89	8965	NULL	NULL	512	x16
Micron	p33	28f512p33e	89	899e	NULL	NULL	512	x16
Micron	p33	28f512p33t	89	8964	NULL	NULL	512	x16
Micron	p33	28f640p33b	89	8820	NULL	NULL	64	x16
Micron	p33	28f640p33t	89	881d	NULL	NULL	64	x16

表 50：**支持用于 Virtex 7 SPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	128	x1、x2 和 x4
GigaDevice	gd25l	gd25lq256d	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	1024	x1、x2 和 x4
ISSI	is25lp	is25lp016d	16	x1、x2 和 x4
ISSI	is25lp	is25lp032d	32	x1、x2 和 x4
ISSI	is25lp	is25lp064a	64	x1、x2 和 x4
ISSI	is25lp	is25lp080d	8	x1、x2 和 x4
ISSI	is25lp	is25lp128f	128	x1、x2 和 x4
ISSI	is25lp	is25lp256d	256	x1、x2 和 x4
ISSI	is25lp	is25lp512m	512	x1、x2 和 x4
ISSI	is25w	is25wp01g	1024	x1、x2 和 x4
ISSI	is25wp	is25wp016d	16	x1、x2 和 x4
ISSI	is25wp	is25wp032d	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	256	x1、x2 和 x4
ISSI	is25wp	is25wp512m	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	512	x1、x2 和 x4
Infineon	s25flxxxk	s25fl116k	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	128	x1、x2 和 x4
Infineon	s25flxxxl	s25fl256l	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	32	x1、x2 和 x4

表 50：**支持用于 Virtex 7 SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	密度 (Mbit)	数据位宽
Infineon	s25flxxxp	s25fl064p	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4]	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx1	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	512	x1、x2 和 x4
Infineon	s25hlt	s25hl01gt	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	512	x1、x2 和 x4
Infineon	s25hst	s25hs01gt	1024	x1、x2 和 x4
Infineon	s25hst	s25hs02gt	2048	x1、x2 和 x4
Infineon	s25hst	s25hs512t	512	x1、x2 和 x4
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4]	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4]	256	x1、x2 和 x4
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4]	32	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4]	512	x1、x2 和 x4
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4]	64	x1、x2 和 x4
Macronix	mx25l	mx25v1635f	16	x1、x2 和 x4
Macronix	mx25l	mx25v8035f	8	x1、x2 和 x4
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4]	128	x1、x2 和 x4
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4]	16	x1、x2 和 x4

表 50：**支持用于 Virtex 7 SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4]	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4]	32	x1、x2 和 x4
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4]	512	x1、x2 和 x4
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4]	64	x1、x2 和 x4
Macronix	mx25u	mx25u8035f [mx25u8033e-spi-x1_x2_x4]	8	x1、x2 和 x4
Macronix	mx66l	mx66l1g45g	1024	x1、x2 和 x4
Macronix	mx66l	mx66l2g45g	2048	x1、x2 和 x4
Macronix	mx66u	mx66u1g45g	1024	x1、x2 和 x4
Macronix	mx66u	mx66u2g45g	2048	x1、x2 和 x4
Micron	mt25ql	mt25ql01g	1024	x1、x2 和 x4
Micron	mt25ql	mt25ql02g	2048	x1、x2 和 x4
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	128	x1、x2 和 x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	256	x1、x2 和 x4
Micron	mt25ql	mt25ql512	512	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	128	x1、x2 和 x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	256	x1、x2 和 x4
Micron	mt25qu	mt25qu512	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	32	x1、x2 和 x4
Micron	n25q	n25q32-3.3v	32	x1、x2 和 x4

表 50：**支持用于 Virtex 7 SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	密度 (Mbit)	数据位宽
Micron	n25q	n25q64-1.8v	64	x1、x2 和 x4
Micron	n25q	n25q64-3.3v	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	128	x1、x2 和 x4
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	128	x1、x2 和 x4

Artix UltraScale+ 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 Artix UltraScale+ 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 51：**支持用于 Artix UltraScale+ BPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	2223	2201	512	x16 和 x8
Infineon	s29glxxxs	s29gl01gs	1	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	2222	2201	256	x16 和 x8
Micron	g18	28f128g18f	89	NULL	NULL	128	x16

表 51：**支持用于 Artix UltraScale+ BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	2223	2201	512	x16 和 x8
Micron	mt28fw	mt28fw02gb	89	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	NULL	NULL	1024	x16

表 51：**支持用于 Artix UltraScale+ BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	p30	28f00ap30t	89	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	NULL	NULL	64	x16

表 52：**支持用于 Artix UltraScale+ SPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2、x4 和 x8
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2、x4 和 x8
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2、x4 和 x8
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2、x4 和 x8
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2、x4 和 x8

表 52：**支持用于 Artix UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25fl	s25fl02g		NULL	NULL	2048	x1、x2、x4 和 x8
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2、x4 和 x8
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2、x4 和 x8
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2、x4 和 x8
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2、x4 和 x8

表 52：**支持用于 Artix UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4, mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4_x8]	c2	20	18	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4, mx25l25635f-mx25l25645g-spi-x1_x2_x4_x8]	c2	20	19	256	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4, mx25l51245g-mx66l51235f-spi-x1_x2_x4_x8]	c2	20	1a	512	x1、x2 和 x4
Macronix	mx25lu	mx25u8035f [mx25u8033e-spi-x1_x2_x4、mx25u8033e-spi-x1_x2_x4_x8]	c2	25	34	8	x1、x2、x4 和 x8
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4, mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4_x8]	c2	25	38	128	x1、x2、x4 和 x8

表 52：**支持用于 Artix UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4、mx25u1632f-spi-x1_x2_x4_x8]	c2	25	35	16	x1、x2 和 x4
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	c2	25	39	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4、mx25u3232f-spi-x1_x2_x4_x8]	c2	25	36	32	x1、x2 和 x4
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4、mx66u51235f-spi-x1_x2_x4_x8]	c2	25	3a	512	x1、x2、x4 和 x8
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	c2	25	37	64	x1、x2、x4 和 x8
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2、x4 和 x8
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2、x4 和 x8

表 52：**支持用于 Artix UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2、x4 和 x8
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4
Winbond	w25q	w25q256jw	ef	60	19	256	x1、x2、x4 和 x8

Kintex UltraScale 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 Kintex UltraScale 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 53：**支持用于 Kintex UltraScale BPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	227e	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	227e	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxp	s70gl02gp	1	227e	2248	2201	2048	x16
Infineon	s29glxxxs	s29gl01gs	1	227e	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	227e	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	227e	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	227e	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	227e	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	227e	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	227e	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	227e	2222	2201	256	x16 和 x8

表 53：**支持用于 Kintex UltraScale BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	g18	28f128g18f	89	8900	NULL	NULL	128	x16
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	88b0	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	8901	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	887e	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	227e	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	227e	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	227e	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	227e	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	227e	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	227e	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	227e	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	227e	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	227e	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w640gh	20	227e	220c	2201	64	x16 和 x8
Micron	m29w	m29w640gl	20	227e	220c	2200	64	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	227e	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	227e	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	227e	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	227e	2223	2201	512	x16 和 x8

表 53：**支持用于 Kintex UltraScale BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	mt28fw	mt28fw02gb	89	227e	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	8963	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	899a	NULL	NULL	1024	x16
Micron	p30	28f00ap30t	89	8962	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	899a	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	881b	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	8818	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	891c	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	8919	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	8961	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	8999	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	8960	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	881a	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	8817	NULL	NULL	64	x16
Micron	p33	28f00ap33b	89	8967	NULL	NULL	1024	x16
Micron	p33	28f00ap33e	89	899f	NULL	NULL	1024	x16
Micron	p33	28f00ap33t	89	8966	NULL	NULL	1024	x16
Micron	p33	28f128p33b	89	8821	NULL	NULL	128	x16
Micron	p33	28f128p33t	89	881e	NULL	NULL	128	x16
Micron	p33	28f256p33b	89	8922	NULL	NULL	256	x16
Micron	p33	28f256p33t	89	891f	NULL	NULL	256	x16
Micron	p33	28f512p33b	89	8965	NULL	NULL	512	x16
Micron	p33	28f512p33e	89	899e	NULL	NULL	512	x16
Micron	p33	28f512p33t	89	8964	NULL	NULL	512	x16
Micron	p33	28f640p33b	89	8820	NULL	NULL	64	x16
Micron	p33	28f640p33t	89	881d	NULL	NULL	64	x16

表 54：**支持用于 Kintex UltraScale SPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2、x4 和 x8
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2、x4 和 x8
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2、x4 和 x8
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2、x4 和 x8
ISSI	is25lp	is25lp016d	9d	60	15	16	x1、x2、x4 和 x8
ISSI	is25lp	is25lp032d	9d	60	16	32	x1、x2 和 x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1、x2 和 x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1、x2 和 x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1、x2 和 x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2、x4 和 x8
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2、x4 和 x8
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2、x4 和 x8
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2、x4 和 x8
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2、x4 和 x8
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2、x4 和 x8
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2、x4 和 x8
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25fl	s25fl02g	-	-	2048	x1、x2、x4 和 x8	
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2、x4 和 x8

表 54：**支持用于 Kintex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2、x4 和 x8
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2、x4 和 x8
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2、x4 和 x8
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2、x4 和 x8
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4, mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4_x8]	c2	20	18	128	x1、x2、x4 和 x8
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4, mx25l25635f-mx25l25645g-spi-x1_x2_x4_x8]	c2	20	19	256	x1、x2、x4 和 x8

表 54：**支持用于 Kintex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4、mx25l3233f-spi-x1_x2_x4_x8]	c2	20	16	32	x1、x2、x4 和 x8
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4, mx25l51245g-mx66l51235f-spi-x1_x2_x4_x8]	c2	20	1a	512	x1、x2、x4 和 x8
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4、mx25l6473f-spi-x1_x2_x4_x8]	c2	20	17	64	x1、x2、x4 和 x8
Macronix	mx25l	mx25v1635f	c2	23	15	16	x1、x2 和 x4
Macronix	mx25l	mx25v8035f	c2	23	14	8	x1、x2、x4 和 x8
Macronix	mx25lu	mx25u8035f [mx25u8033e-spi-x1_x2_x4、mx25u8033e-spi-x1_x2_x4_x8]	c2	25	34	8	x1、x2、x4 和 x8
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4, mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4_x8]	c2	25	38	128	x1、x2、x4 和 x8
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4、mx25u1632f-spi-x1_x2_x4_x8]	c2	25	35	16	x1、x2、x4 和 x8

表 54：**支持用于 Kintex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	c2	25	39	256	x1、x2、x4 和 x8
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4、mx25u3232f-spi-x1_x2_x4_x8]	c2	25	36	32	x1、x2、x4 和 x8
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4、mx66u51235f-spi-x1_x2_x4_x8]	c2	25	3a	512	x1、x2、x4 和 x8
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	c2	25	37	64	x1、x2、x4 和 x8
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1、x2 和 x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1、x2、x4 和 x8
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2、x4 和 x8
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2、x4 和 x8
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1、x2、x4 和 x8
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1、x2、x4 和 x8
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4、n25q128-3.3v-spi-x1_x2_x4_x8]	20	ba	18	128	x1、x2 和 x4

表 54：**支持用于 Kintex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4、n25q256-3.3v-spi-x1_x2_x4_x8]	20	ba	19	256	x1、x2 和 x4
Micron	mt25ql	mt25ql512	20	ba	20	512	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1、x2 和 x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2、x4 和 x8
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4
Winbond	w25q	w25q256jw	ef	60	19	256	x1、x2、x4 和 x8

Kintex UltraScale+ 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 Kintex UltraScale+ 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 55：**支持用于 Kintex UltraScale+ BPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	2223	2201	512	x16 和 x8
Infineon	s29glxxxs	s29gl01gs	1	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	2222	2201	256	x16 和 x8
Micron	g18	28f128g18f	89	NULL	NULL	128	x16

表 55：**支持用于 Kintex UltraScale+ BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	2223	2201	512	x16 和 x8
Micron	mt28fw	mt28fw02gb	89	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	NULL	NULL	1024	x16

表 55：**支持用于 Kintex UltraScale+ BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	p30	28f00ap30t	89	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	NULL	NULL	64	x16

表 56：**支持用于 Kintex UltraScale+ SPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2、x4 和 x8
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2、x4 和 x8
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2、x4 和 x8
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2、x4 和 x8
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2、x4 和 x8

表 56：**支持用于 Kintex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25fl	s25fl02g		NULL	NULL	2048	x1、x2、x4 和 x8
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2、x4 和 x8
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2、x4 和 x8
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2、x4 和 x8
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2、x4 和 x8

表 56：**支持用于 Kintex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4, mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4_x8]	c2	20	18	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4, mx25l25635f-mx25l25645g-spi-x1_x2_x4_x8]	c2	20	19	256	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4, mx25l51245g-mx66l51235f-spi-x1_x2_x4_x8]	c2	20	1a	512	x1、x2 和 x4
Macronix	mx25lu	mx25u8035f [mx25u8033e-spi-x1_x2_x4、mx25u8033e-spi-x1_x2_x4_x8]	c2	25	34	8	x1、x2、x4 和 x8
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4, mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4_x8]	c2	25	38	128	x1、x2、x4 和 x8

表 56：**支持用于 Kintex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4、mx25u1632f-spi-x1_x2_x4_x8]	c2	25	35	16	x1、x2 和 x4
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	c2	25	39	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4、mx25u3232f-spi-x1_x2_x4_x8]	c2	25	36	32	x1、x2 和 x4
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4、mx66u51235f-spi-x1_x2_x4_x8]	c2	25	3a	512	x1、x2、x4 和 x8
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	c2	25	37	64	x1、x2、x4 和 x8
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2、x4 和 x8
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2、x4 和 x8

表 56：**支持用于 Kintex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2、x4 和 x8
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4
Winbond	w25q	w25q256jw	ef	60	19	256	x1、x2、x4 和 x8

Virtex UltraScale 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 AMD Virtex™ UltraScale™ 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 57：**支持用于 Virtex UltraScale BPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	227e	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	227e	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxp	s70gl02gp	1	227e	2248	2201	2048	x16
Infineon	s29glxxxs	s29gl01gs	1	227e	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	227e	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	227e	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	227e	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	227e	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	227e	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	227e	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	227e	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	227e	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	227e	2222	2201	256	x16 和 x8

表 57：**支持用于 Virtex UltraScale BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	g18	28f128g18f	89	8900	NULL	NULL	128	x16
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	88b0	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	8901	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	887e	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	227e	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	227e	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	227e	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	227e	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	227e	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	227e	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	227e	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	227e	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	227e	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	227e	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	227e	2222	2201	256	x16 和 x8
Micron	m29w	m29w640gh	20	227e	220c	2201	64	x16 和 x8
Micron	m29w	m29w640gl	20	227e	220c	2200	64	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	227e	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	227e	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	227e	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	227e	2223	2201	512	x16 和 x8

表 57：**支持用于 Virtex UltraScale BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	DEVICE_ID_1	DEVICE_ID_2	密度 (Mbit)	数据位宽
Micron	mt28fw	mt28fw02gb	89	227e	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	8963	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	899a	NULL	NULL	1024	x16
Micron	p30	28f00ap30t	89	8962	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	899a	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	881b	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	8818	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	891c	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	8919	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	8961	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	8999	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	8960	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	881a	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	8817	NULL	NULL	64	x16
Micron	p33	28f00ap33b	89	8967	NULL	NULL	1024	x16
Micron	p33	28f00ap33e	89	899f	NULL	NULL	1024	x16
Micron	p33	28f00ap33t	89	8966	NULL	NULL	1024	x16
Micron	p33	28f128p33b	89	8821	NULL	NULL	128	x16
Micron	p33	28f128p33t	89	881e	NULL	NULL	128	x16
Micron	p33	28f256p33b	89	8922	NULL	NULL	256	x16
Micron	p33	28f256p33t	89	891f	NULL	NULL	256	x16
Micron	p33	28f512p33b	89	8965	NULL	NULL	512	x16
Micron	p33	28f512p33e	89	899e	NULL	NULL	512	x16
Micron	p33	28f512p33t	89	8964	NULL	NULL	512	x16
Micron	p33	28f640p33b	89	8820	NULL	NULL	64	x16
Micron	p33	28f640p33t	89	881d	NULL	NULL	64	x16

表 58：**支持用于 Virtex UltraScale SPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2、x4 和 x8
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2、x4 和 x8
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2、x4 和 x8
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2、x4 和 x8
ISSI	is25lp	is25lp016d	9d	60	15	16	x1、x2、x4 和 x8
ISSI	is25lp	is25lp032d	9d	60	16	32	x1、x2 和 x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1、x2 和 x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1、x2 和 x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1、x2 和 x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2、x4 和 x8
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2、x4 和 x8
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2、x4 和 x8
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2、x4 和 x8
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2、x4 和 x8
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2、x4 和 x8
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2、x4 和 x8
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25fl	s25fl02g	-	-	2048	x1、x2、x4 和 x8	
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2、x4 和 x8

表 58：**支持用于 Virtex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2、x4 和 x8
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2、x4 和 x8
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2、x4 和 x8
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2、x4 和 x8
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4, mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4_x8]	c2	20	18	128	x1、x2、x4 和 x8
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4, mx25l25635f-mx25l25645g-spi-x1_x2_x4_x8]	c2	20	19	256	x1、x2、x4 和 x8

表 58：**支持用于 Virtex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4、mx25l3233f-spi-x1_x2_x4_x8]	c2	20	16	32	x1、x2、x4 和 x8
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4, mx25l51245g-mx66l51235f-spi-x1_x2_x4_x8]	c2	20	1a	512	x1、x2、x4 和 x8
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4、mx25l6473f-spi-x1_x2_x4_x8]	c2	20	17	64	x1、x2、x4 和 x8
Macronix	mx25l	mx25v1635f	c2	23	15	16	x1、x2 和 x4
Macronix	mx25l	mx25v8035f	c2	23	14	8	x1、x2、x4 和 x8
Macronix	mx25lu	mx25u8035f [mx25u8033e-spi-x1_x2_x4、mx25u8033e-spi-x1_x2_x4_x8]	c2	25	34	8	x1、x2、x4 和 x8
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4, mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4_x8]	c2	25	38	128	x1、x2、x4 和 x8
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4、mx25u1632f-spi-x1_x2_x4_x8]	c2	25	35	16	x1、x2、x4 和 x8

表 58：**支持用于 Virtex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	c2	25	39	256	x1、x2、x4 和 x8
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4、mx25u3232f-spi-x1_x2_x4_x8]	c2	25	36	32	x1、x2、x4 和 x8
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4、mx66u51235f-spi-x1_x2_x4_x8]	c2	25	3a	512	x1、x2、x4 和 x8
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	c2	25	37	64	x1、x2、x4 和 x8
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1、x2 和 x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1、x2、x4 和 x8
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2、x4 和 x8
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2、x4 和 x8
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1、x2、x4 和 x8
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1、x2、x4 和 x8
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4、n25q128-3.3v-spi-x1_x2_x4_x8]	20	ba	18	128	x1、x2 和 x4

表 58：**支持用于 Virtex UltraScale SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	MANUFACTURER_READ_ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4、n25q256-3.3v-spi-x1_x2_x4_x8]	20	ba	19	256	x1、x2 和 x4
Micron	mt25ql	mt25ql512	20	ba	20	512	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1、x2 和 x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2、x4 和 x8
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4
Winbond	w25q	w25q256jw	ef	60	19	256	x1、x2、x4 和 x8

Virtex UltraScale+ 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 AMD Virtex™ UltraScale+™ 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 59：**支持用于 Virtex UltraScale+ BPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Infineon	s29glxxxp	s29gl01gp	1	2228	2201	1024	x16 和 x8
Infineon	s29glxxxp	s29gl128p	1	2221	2201	128	x16 和 x8
Infineon	s29glxxxp	s29gl256p	1	2222	2201	256	x16 和 x8
Infineon	s29glxxxp	s29gl512p	1	2223	2201	512	x16 和 x8
Infineon	s29glxxxs	s29gl01gs	1	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	2228	2201	1024	x16 和 x8
Infineon	s29glxxxt	s29gl512t	1	2223	2201	512	x16 和 x8
Infineon	s29glxxxt	s70gl02gt	1	2248	2201	2048	x16 和 x8
Macronix	mx29gl	mx29gl128f	c2	2221	2201	128	x16 和 x8
Macronix	mx29gl	mx29gl256f	c2	2222	2201	256	x16 和 x8
Micron	g18	28f128g18f	89	NULL	NULL	128	x16

表 59：**支持用于 Virtex UltraScale+ BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	89	NULL	NULL	1024	x16
Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	89	NULL	NULL	256	x16
Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	89	NULL	NULL	512	x16
Micron	m29ew	28f00am29ew	89	2228	2201	1024	x16 和 x8
Micron	m29ew	28f00bm29ew	89	2248	2201	2048	x16 和 x8
Micron	m29ew	28f064m29ewb	89	2210	2200	64	x16 和 x8
Micron	m29ew	28f064m29ewh	89	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewl	89	220c	2201	64	x16 和 x8
Micron	m29ew	28f064m29ewt	89	2210	2201	64	x16 和 x8
Micron	m29ew	28f128m29ew	89	2221	2201	128	x16 和 x8
Micron	m29ew	28f256m29ew	89	2222	2201	256	x16 和 x8
Micron	m29ew	28f512m29ew	89	2223	2201	512	x16 和 x8
Micron	m29w	m29w128gh	20	2221	2201	128	x16 和 x8
Micron	m29w	m29w128gl	20	2221	2200	128	x16 和 x8
Micron	m29w	m29w256gh	20	2222	2201	256	x16 和 x8
Micron	m29w	m29w256gl	20	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew01ga	89	2228	2201	1024	x16 和 x8
Micron	mt28ew	mt28ew128a	89	2221	2201	128	x16 和 x8
Micron	mt28ew	mt28ew256a	89	2222	2201	256	x16 和 x8
Micron	mt28ew	mt28ew512a	89	2223	2201	512	x16 和 x8
Micron	mt28fw	mt28fw02gb	89	2248	2201	2048	x16
Micron	p30	28f00ap30b	89	NULL	NULL	1024	x16
Micron	p30	28f00ap30e	89	NULL	NULL	1024	x16

表 59：**支持用于 Virtex UltraScale+ BPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	MEMORY_TYPE_ID	MEMORY_CAPACITY_ID	密度 (Mbit)	数据位宽
Micron	p30	28f00ap30t	89	NULL	NULL	1024	x16
Micron	p30	28f00bp30e	89	NULL	NULL	2048	x16
Micron	p30	28f128p30b	89	NULL	NULL	128	x16
Micron	p30	28f128p30t	89	NULL	NULL	128	x16
Micron	p30	28f256p30b	89	NULL	NULL	256	x16
Micron	p30	28f256p30t	89	NULL	NULL	256	x16
Micron	p30	28f512p30b	89	NULL	NULL	512	x16
Micron	p30	28f512p30e	89	NULL	NULL	512	x16
Micron	p30	28f512p30t	89	NULL	NULL	512	x16
Micron	p30	28f640p30b	89	NULL	NULL	64	x16
Micron	p30	28f640p30t	89	NULL	NULL	64	x16

表 60：**支持用于 Virtex UltraScale+ SPI 器件配置的闪存器件**

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
GigaDevice	gd25f	gd25f128f	c8	43	18	128	x1、x2、x4 和 x8
GigaDevice	gd25l	gd25lq256d	c8	60	19	256	x1、x2 和 x4
GigaDevice	gd25q	gd25q256d	c8	40	19	256	x1、x2 和 x4
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1、x2 和 x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1、x2、x4 和 x8
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1、x2、x4 和 x8
ISSI	is25wp	is25wp016d	9d	70	15	16	x1、x2、x4 和 x8
ISSI	is25wp	is25wp032d	9d	70	16	32	x1、x2 和 x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1、x2 和 x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1、x2 和 x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1、x2 和 x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1、x2、x4 和 x8

表 60：**支持用于 Virtex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1、x2 和 x4
ISSI	is25wp	is25wp512mg	9d	70	20	512	x1、x2 和 x4
Infineon	s25fl	s25fl02g		NULL	NULL	2048	x1、x2、x4 和 x8
Infineon	s25flxxxk	s25fl116k	1	40	15	16	x1、x2 和 x4
Infineon	s25flxxxk	s25fl132k	1	40	16	32	x1、x2 和 x4
Infineon	s25flxxxk	s25fl164k	1	40	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1、x2 和 x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1、x2、x4 和 x8
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1、x2 和 x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1、x2 和 x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl128sxxxxxx1	1	20	18	128	x1、x2 和 x4
Infineon	s25flxxxs	s25fl256sxxxxxx0	1	2	19	256	x1、x2、x4 和 x8
Infineon	s25flxxxs	s25fl256sxxxxxx1	1	2	19	256	x1、x2 和 x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1、x2、x4 和 x8
Infineon	s25hlt	s25hl01gt	34	2a	1b	1024	x1、x2 和 x4
Infineon	s25hlt	s25hl02gt	34	2a	1c	2048	x1、x2 和 x4
Infineon	s25hlt	s25hl512t	34	2a	1a	512	x1、x2、x4 和 x8
Infineon	s25hst	s25hs01gt	34	2b	1b	1024	x1、x2、x4 和 x8
Infineon	s25hst	s25hs02gt	34	2b	1c	2048	x1、x2、x4 和 x8
Infineon	s25hst	s25hs512t	34	2b	1a	512	x1、x2、x4 和 x8

表 60：**支持用于 Virtex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4, mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4_x8]	c2	20	18	128	x1、x2 和 x4
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4, mx25l25635f-mx25l25645g-spi-x1_x2_x4_x8]	c2	20	19	256	x1、x2 和 x4
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4, mx25l51245g-mx66l51235f-spi-x1_x2_x4_x8]	c2	20	1a	512	x1、x2 和 x4
Macronix	mx25lu	mx25u8035f [mx25u8033e-spi-x1_x2_x4、mx25u8033e-spi-x1_x2_x4_x8]	c2	25	34	8	x1、x2、x4 和 x8
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4, mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4_x8]	c2	25	38	128	x1、x2、x4 和 x8

表 60：**支持用于 Virtex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4、mx25u1632f-spi-x1_x2_x4_x8]	c2	25	35	16	x1、x2 和 x4
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	c2	25	39	256	x1、x2 和 x4
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4、mx25u3232f-spi-x1_x2_x4_x8]	c2	25	36	32	x1、x2 和 x4
Macronix	mx25u	mx25u51245gxxj [mx66u51235f-spi-x1_x2_x4、mx66u51235f-spi-x1_x2_x4_x8]	c2	25	3a	512	x1、x2、x4 和 x8
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	c2	25	37	64	x1、x2、x4 和 x8
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1、x2、x4 和 x8
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1、x2 和 x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1、x2 和 x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1、x2、x4 和 x8

表 60：**支持用于 Virtex UltraScale+ SPI 器件配置的闪存器件** (续)

制造商	制造商家族	器件别名	制造商 ID	器件 ID（十六进制）	器件 ID（十六进制）	密度 (Mbit)	数据位宽
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256	x1、x2、x4 和 x8
Micron	mt25qu	mt25qu512	20	bb	20	512	x1、x2 和 x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1、x2 和 x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1、x2 和 x4
Winbond	w25q	w25q128jv	EF	40	18	128	x1、x2 和 x4
Winbond	w25q	w25q128jvm	EF	70	18	128	x1、x2、x4 和 x8
Winbond	w25q	w25q12pw [w25q128jw, w25q128jwm, w25q128jwm]	EF	60	18	128	x1、x2 和 x4
Winbond	w25q	w25q12pwm [w25q128jwm, w25q128jwm]	EF	80	18	128	x1、x2 和 x4
Winbond	w25q	w25q256jw	ef	60	19	256	x1、x2、x4 和 x8

Zynq 7000 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 Zynq 7000 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

注释：用于构建“Configuration Memory Device Programmer”（配置存储器器件烧录器）的 U-Boot 标签如下所示：

接口	U-Boot 标签
qspi	xilinx-v2024.01
nor	xilinx-v2024.01
nand	xilinx-v2024.01

表 61：**支持用于 Zynq 7000 器件配置的闪存器件**

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
NAND	Infineon	s34ml	s34ml01g1	1024	x16 和 x8
NAND	Infineon	s34ml	s34ml02g1	2048	x16 和 x8
NAND	Micron	mt29f	mt29f2g08ab	2048	x8
NAND	Micron	mt29f	mt29f2g16ab	2048	x16
NOR	Micron	m29ew	28f032m29ewt	32	x8
NOR	Micron	m29ew	28f064m29ewt	64	x8
NOR	Micron	m29ew	28f128m29ewh	128	x8
NOR	Micron	m29ew	28f256m29ewh	256	x8
NOR	Micron	m29ew	28f512m29ewh	512	x8

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25l	is25lp01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp02gg	2047	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25lp	is25lp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lx	is25lx512m	509	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25w	is25wp01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25wp	is25wp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256e	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxl	s25fl064l	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxl	s25fl256l	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxp	s25fl129p	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-1.8v	128	x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-3.3v	128	x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl128s-1.8v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Infineon	s25flxxxs	s25fl128s-3.3v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s70flxxxs	s70fl01gs_00	1024	x4-dual_stacked
QSPI	Macronix	mx25	MX25U25643G	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12833f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25l	mx25l12835f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12845g	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12872f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25635f	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25645g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25673g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l3273f	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25l	mx25l51273g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12832f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12843g	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12872f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25635f	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25645g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25673g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25u	mx25u51245gxxj	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u6472f	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u8035f	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l51235f	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u1g45g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u2g45g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25ql	mt25ql01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql02g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql512	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q00a-3.3v	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q128-3.3v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q256-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25ql	n25q512-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu512	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q00a-1.8v	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q128-1.8v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25qu	n25q256-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q512-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25h	w25h02jv	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q02nw	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fv	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fw	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 61：**支持用于 Zynq 7000 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Winbond	w25q	w25q128jv	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

注释：不支持 Macronix MX66U2G45G_54 子家族。

Zynq UltraScale+ MPSoC 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 Zynq UltraScale+ MPSoC 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

注释：不支持 Macronix MX66U2G45G_54 子家族。

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件**

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
EMMC	不适用		jedec4.51	131072	
EMMC	不适用		jedec4.51	262144	
EMMC	不适用		jedec4.51	32768	
EMMC	不适用		jedec4.51	524288	
EMMC	不适用		jedec4.51	65536	
NAND	Infineon	s34ml	s34ml01g1	1024	x8-dual、x8-single
NAND	Infineon	s34ml	s34ml02g1	2048	x8-dual、x8-single
NAND	Micron	mt29f	mt29f16g08ab	16384	x8-dual、x8-single
NAND	Micron	mt29f	mt29f2g08ab	2048	x8-dual、x8-single
NAND	Micron	mt29f	mt29f32g08ae	32768	x8-dual、x8-single
NAND	Micron	mt29f	mt29f64g08ae	65536	x8-dual、x8-single
NAND	Micron	mt29f	mt29f8g08ab	8192	x8-dual、x8-single

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	GigaDevice	gd25b	gd25b512me	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25l	is25lp01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp01gg	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp02gg	2047	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25lp	is25lp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lx	is25lx512m	509	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25w	is25wp01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25wp	is25wp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256e	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxl	s25fl256l	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxp	s25fl129p	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-1.8v	128	x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-3.3v	128	x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl128s-1.8v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Infineon	s25flxxxs	s25fl128s-3.3v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s70flxxxs	s70fl01gs_00	1024	x4-dual_stacked
QSPI	Macronix	mx25	MX25U25643G	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12833f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25l	mx25l12835f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12872f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25635f	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25645g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25673g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l3273f	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51273g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25u	mx25u12832f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12843g	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12872f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25635f	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25645g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25673g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u51245gxxj	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25u	mx25u6472f	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u8035f	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u1g45g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u2g45g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u51235f	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25ql	mt25ql02g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql512	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q00a-3.3v	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q128-3.3v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q256-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q512-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25qu	mt25qu01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu512	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q00a-1.8v	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q128-1.8v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q256-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25qu	n25q512-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	不适用		qspi	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25h	w25h02jv	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q02nw	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fw	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q256jw	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 62：**支持用于 Zynq UltraScale+ MPSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Winbond	w25q	w25q256jwm	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

Zynq UltraScale+ RFSoC 配置存储器器件

下表所示闪存器件支持通过 AMD Vivado™ 软件对 Zynq UltraScale+ RFSoC 器件执行擦除、空白检查、烧录和验证等配置操作。

本附录中的表格所列 AMD 家族非易失性存储器将不断保持更新，并支持通过 Vivado 软件对其中所列非易失性存储器进行擦除、空白检查、烧录和验证。AMD 竭尽所能保留此列表上的组件，即使这些组件不再适用于新设计也是如此，从而为可能包含这些组件的最终产品提供长期维护支持。

重要提示！鉴于商用非易失性存储器市场不断演变，AMD 建议您与自己的非易失性存储器供应商联系，以确认器件可用性和生命周期。下表中引用的特定器件并不能作为其当前或未来可用性的保证。

重要提示！由 Spansion 制造的闪存器件现在被称为 Infineon。只要部件号相同，就没有功能差异。

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件**

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
EMMC	Micron	mtfc	mtfc8gakajcn	65536	
EMMC	不适用		jedec4.51-emmc	524288	
EMMC	不适用		jedec4.51	131072	
EMMC	不适用		jedec4.51	262144	
EMMC	不适用		jedec4.51	32768	
EMMC	不适用		jedec4.51	524288	
EMMC	不适用		jedec4.51	65536	
NAND	Infineon	s34ml	s34ml01g1	1024	x8-dual、x8-single
NAND	Infineon	s34ml	s34ml02g1	2048	x8-dual、x8-single
NAND	Micron	mt29f	mt29f16g08ab	16384	x8-dual、x8-single
NAND	Micron	mt29f	mt29f2g08ab	2048	x8-dual、x8-single
NAND	Micron	mt29f	mt29f32g08ae	32768	x8-dual、x8-single
NAND	Micron	mt29f	mt29f64g08ae	65536	x8-dual、x8-single
NAND	Micron	mt29f	mt29f8g08ab	8192	x8-dual、x8-single

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	GigaDevice	gd25b	gd25b512me	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25l	is25lp01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp01gg	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp02gg	2047	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25lp	is25lp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lx	is25lx512m	509	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25w	is25wp01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	ISSI	is25wp	is25wp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256e	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxl	s25fl256l	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxp	s25fl129p	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-1.8v	128	x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-3.3v	128	x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl128s-1.8v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Infineon	s25flxxxs	s25fl128s-3.3v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s70flxxxs	s70fl01gs_00	1024	x4-dual_stacked
QSPI	Macronix	mx25	MX25U25643G	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12833f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25l	mx25l12835f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12872f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25635f	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25645g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25673g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l3273f	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51273g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25u	mx25u12832f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12843g	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12872f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25635f	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25645g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25673g	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u51245gxxj	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Macronix	mx25u	mx25u6472f	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u8035f	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u1g45g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u2g45g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u51235f	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25ql	mt25ql02g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql512	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q00a-3.3v	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q128-3.3v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q256-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	n25q512-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25qu	mt25qu01g	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu512	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q00a-1.8v	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q128-1.8v	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	n25q256-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Micron	mt25qu	n25q512-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	不适用		qspi	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25h	w25h02jv	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q02nw	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fw	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q256jw	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 63：**支持用于 Zynq UltraScale+ RFSoC 器件配置的闪存器件** (续)

接口	制造商	制造商家族	器件	密度 (Mbit)	数据位宽
QSPI	Winbond	w25q	w25q256jwm	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

注释：不支持 Macronix MX66U2G45G_54 子家族。

Versal 配置存储器器件

注释：对于具有正式的业界规范的闪存器件（如 eMMC 和 SD），我们遵循相应的规范版本来提供器件支持，而不是通过枚举每个特定制造商、制造商家族和器件的部件号来提供支持。这样即可在初始闪存选择以及长期设计维护方面为设计师提供广泛的可能性。

注释：不支持 Macronix MX66U2G45G_54 子家族。

表 64：**Versal 配置存储器器件**

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
EMMC	不适用	jedec4.51-16gb	jedec4.51-16gb	131072		
EMMC	不适用	jedec4.51-32gb	jedec4.51-32gb	262144		
EMMC	不适用	jedec4.51-4gb	jedec4.51-4gb	32768		
EMMC	不适用	jedec4.51-64gb	jedec4.51-64gb	524288		
EMMC	不适用	jedec4.51-8gb	jedec4.51-8gb	65536		
OSPI	GigaDevice	gd25	gd25lx512m	cfgmem-<flash Density>-ospi-<data_width_bits>	512	x8-dual_stacked、x8-single
OSPI	GigaDevice	gd25lx	gd25lx256e	cfgmem-<flash Density>-ospi-<data_width_bits>	256	x8-dual_stacked、x8-single
OSPI	GigaDevice	gd55	gd55lx01g	cfgmem-<flash Density>-ospi-<data_width_bits>	1024	x8-dual_stacked、x8-single
OSPI	GigaDevice	gd55	gd55lx02g	cfgmem-<flash Density>-ospi-<data_width_bits>	2048	x8-dual_stacked、x8-single
OSPI	Infineon	s28hst	s28hs02gt	cfgmem-<flash Density>-ospi-<data_width_bits>	2048	x8-dual_stacked、x8-single
OSPI	Macronix	mx25u	mx25um51345g	cfgmem-<flash Density>-ospi-<data_width_bits>	512	x8-dual_stacked、x8-single
OSPI	Macronix	mx66um	mx66um2g45g	cfgmem-<flash Density>-ospi-<data_width_bits>	2048	x8-dual_stacked、x8-single

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
OSPI	Micron	mt35	mt35xu02gcba1g12-osit	cfgmem-<flash Density>-ospi-<data_width_bits>	2048	
QSPI	GigaDevice	gd25b	gd25b512me	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25l	is25lp01g	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	cfgmem-<flash Density>-qspi-<data_width_bits>	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp01gg	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp02gg	cfgmem-<flash Density>-qspi-<data_width_bits>	2047	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	ISSI	is25lp	is25lp032d	cfgmem-<flash Density>-qspi-<data_width_bits>	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp064a	cfgmem-<flash Density>-qspi-<data_width_bits>	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp080d	cfgmem-<flash Density>-qspi-<data_width_bits>	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp128f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp256d	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	ISSI	is25lp	is25lp512m	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lx	is25lx512m	cfgmem-<flash Density>-qspi-<data_width_bits>	509	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25w	is25wp01g	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	cfgmem-<flash Density>-qspi-<data_width_bits>	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp032d	cfgmem-<flash Density>-qspi-<data_width_bits>	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	ISSI	is25wp	is25wp064a	cfgmem-<flash Density>-qspi-<data_width_bits>	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	cfgmem-<flash Density>-qspi-<data_width_bits>	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp128f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256d	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256e	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	ISSI	is25wp	is25wp512m	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxl	s25fl256l	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxp	s25fl129p	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Infineon	s25flxxxs	s25fl127s-1.8v	cfgmem-<flash Density>-qspi-<data_width_bits>	128	
QSPI	Infineon	s25flxxxs	s25fl127s-3.3v	cfgmem-<flash Density>-qspi-<data_width_bits>	128	
QSPI	Infineon	s25flxxxs	s25fl128s-1.8v	cfgmem-<flash Density>-qspi-<data_width_bits>	128	
QSPI	Infineon	s25flxxxs	s25fl128s-3.3v	cfgmem-<flash Density>-qspi-<data_width_bits>	128	
QSPI	Infineon	s25flxxxs	s25fl256s-1.8v	cfgmem-<flash Density>-qspi-<data_width_bits>	256	
QSPI	Infineon	s25flxxxs	s25fl256s-3.3v	cfgmem-<flash Density>-qspi-<data_width_bits>	256	

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Infineon	s25flxxxs	s25fl512s-1.8v	cfgmem-<flash Density>-qspi-<data_width_bits>	512	
QSPI	Infineon	s25flxxxs	s25fl512s-3.3v	cfgmem-<flash Density>-qspi-<data_width_bits>	512	
QSPI	Infineon	s70flxxxs	s70fl01gs_00	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x4-dual_stacked
QSPI	Macronix	mx25	MX25U25643G	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12833f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12835f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12872f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Macronix	mx25l	mx25l25635f	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25645g	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25673g	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l3273f	cfgmem-<flash Density>-qspi-<data_width_bits>	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Macronix	mx25l	mx25l51273g	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12832f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12835f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12843g	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12872f	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Macronix	mx25u	mx25u25635f	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25645g	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25673g	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u51245gxxj	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25u	mx25u6472f	cfgmem-<flash Density>-qspi-<data_width_bits>	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Macronix	mx25u	mx25u8035f	cfgmem-<flash Density>-qspi-<data_width_bits>	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u1g45g	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx66u	mx66u2g45g	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Macronix	mx66u	mx66u51235f	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql01g	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql02g	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Micron	mt25ql	mt25ql512	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu01g	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Micron	mt25qu	mt25qu512	cfgmem-<flash Density>-qspi-<data_width_bits>	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-1.8v	cfgmem-<flash Density>-qspi-<data_width_bits>	64	
QSPI	Micron	n25q	n25q64-3.3v	cfgmem-<flash Density>-qspi-<data_width_bits>	64	
QSPI	Winbond	w25h	w25h02jv	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q02nw	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fw	cfgmem-<flash Density>-qspi-<data_width_bits>	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q256jw	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 64：**Versal 配置存储器器件** (续)

接口	制造商	制造商家族	器件	Vivado 选择	密度 (Mb)	数据位宽
QSPI	Winbond	w25q	w25q256jwm	cfgmem-<flash Density>-qspi-<data_width_bits>	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

供应商确认的闪存器件表

注释：支持的数据位宽：x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel。

重要提示！如需获取有关下表中任何供应商确认的闪存器件的支持，请联系相应的制造商。

表 65：**供应商确认的闪存器件表**

接口	制造商	制造商家族	器件名称	Vivado 选择	密度 (Mb)	支持的架构
QSPI	GigaDevice	GD25B	GD25B16E	cfgmem-<flash Density>-qspi-<data_width_bits>	16	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25B	GD25B32E	cfgmem-<flash Density>-qspi-<data_width_bits>	32	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25B	GD25B64E	cfgmem-<flash Density>-qspi-<data_width_bits>	64	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25B	GD25B128E	cfgmem-<flash Density>-qspi-<data_width_bits>	128	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25B	GD25B256E	cfgmem-<flash Density>-qspi-<data_width_bits>	256	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25B	GD55B01GE	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25B	GD55B02GE	cfgmem-<flash Density>-qspi-<data_width_bits>	2046	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25F	GD25F64F	cfgmem-<flash Density>-qspi-<data_width_bits>	64	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25F	GD25F128F	cfgmem-<flash Density>-qspi-<data_width_bits>	128	Versal、Zynq Ultrascale+

表 65：**供应商确认的闪存器件表** (续)

接口	制造商	制造商家族	器件名称	Vivado 选择	密度 (Mb)	支持的架构
QSPI	GigaDevice	GD25F	GD25F256F	cfgmem-<flash Density>-qspi-<data_width_bits>	256	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25F	GD55F512MF	cfgmem-<flash Density>-qspi-<data_width_bits>	512	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD25LB16E	cfgmem-<flash Density>-qspi-<data_width_bits>	16	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD25LB32E	cfgmem-<flash Density>-qspi-<data_width_bits>	32	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD25LB64E	cfgmem-<flash Density>-qspi-<data_width_bits>	64	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD25LB128E	cfgmem-<flash Density>-qspi-<data_width_bits>	128	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD25LB256E	cfgmem-<flash Density>-qspi-<data_width_bits>	256	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD25LB512ME	cfgmem-<flash Density>-qspi-<data_width_bits>	512	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD55LB01GE	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LB	GD55LB02GE	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LF	GD25LF80E	cfgmem-<flash Density>-qspi-<data_width_bits>	8	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LF	GD25LF16E	cfgmem-<flash Density>-qspi-<data_width_bits>	16	Versal、Zynq Ultrascale+

表 65：**供应商确认的闪存器件表** (续)

接口	制造商	制造商家族	器件名称	Vivado 选择	密度 (Mb)	支持的架构
QSPI	GigaDevice	GD25LF	GD25LF32E	cfgmem-<flash Density>-qspi-<data_width_bits>	32	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LF	GD25LF64E	cfgmem-<flash Density>-qspi-<data_width_bits>	64	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LF	GD25LF128E	cfgmem-<flash Density>-qspi-<data_width_bits>	128	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LF	GD25LF255E	cfgmem-<flash Density>-qspi-<data_width_bits>	256	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LF	GD55LF511ME	cfgmem-<flash Density>-qspi-<data_width_bits>	512	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LT	GD25LT256E	cfgmem-<flash Density>-qspi-<data_width_bits>	256	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LT	GD25LT512ME	cfgmem-<flash Density>-qspi-<data_width_bits>	512	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LT	GD55LT01GE	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25LT	GD55LT02GE	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25T	GD25T512ME	cfgmem-<flash Density>-qspi-<data_width_bits>	512	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25T	GD55T01GE	cfgmem-<flash Density>-qspi-<data_width_bits>	1024	Versal、Zynq Ultrascale+
QSPI	GigaDevice	GD25T	GD55T02GE	cfgmem-<flash Density>-qspi-<data_width_bits>	2048	Versal、Zynq Ultrascale+

表 65：**供应商确认的闪存器件表** (续)

接口	制造商	制造商家族	器件名称	Vivado 选择	密度 (Mb)	支持的架构
QSPI	GigaDevice	GD25B	GD25B512ME	cfgmem-<flash Density>-qspi-<data_width_bits>	256	Versal、Zynq Ultrascale+

附录 G

hw_server 的命令行选项

以下是 hw_server 命令行选项的列表。

重要提示！远程连接到 PC 上的 hw_server 时，请确保正确配置防火墙策略。请确保 `hw_server.exe` 有权监听端口 3121 上的新套接字连接。

标准 hw_server 选项

表 66：**标准 hw_server 选项**

选项	用途	示例
`-d`	以守护程序模式运行 hw_server（输出已发送至系统记录器）。	`hw_server -d`
`--help`	基本命令行帮助。	
`-I`	如果指定时间内未建立目标连接，则退出。此处指定的时间以秒为单位。	`hw_server -I 20`

`-L`	启用从客户端到 hw_server 的 JTAG 命令记录。	`hw_server -L-`
		上述选项会将输出记录到 `stdout`
		`hw_server -Lmy_file.log`
		上述选项会将输出记录到 `my_file.log` 文件

`-s`	设置代理，用于监听端口和协议。	`hw_server -stcp::3122` 上述选项用于在 hw_server 启动时与端口 3122 建立连接（使用本地主机）。
`-q`	在启动时不显示版本信息。	

-p	指定供 AMD GDB 服务器使用的端口范围，或者禁用该功能。	`hw_server -p<port>`
		`<port>` 表示 GDB 服务器的基本端口号。默认值为 3000
		服务器会根据目标架构打开 1 个对应端口：
		3000：Arm®；
		3001：Arm64；
		3002：MicroBlaze™；
		3003：MicroBlaze64
		`hw_server -p0 disables the port`
		3004：RISC-V 32-BIT；
		3005：RISC-V 64-BIT

表 66：**标准 hw_server 选项** (续)

选项	用途	示例
`--init`	该选项用于指定到 hw_server 的初始化脚本文件	`hw_server --init my_init.txt` `my_init.txt` 文件包含要在启动时应用于 hw_server 的选项

	环境变量 `HW_SERVER_INIT_FILE`
	用于指定默认 `--init` 文件
		`export HW_SERVER_INIT_FILE=~/my_init.txt`
		`hw_server`
		在此情况下，使用来自环境变量 `HW_SERVER_INIT_FILE` 指向的文件的设置对 hw_server 进行初始化。

高级选项

表 67：**高级选项**

选项	用途

`detect-ir-length`	禁用在扫描链发现期间对 IR 检测长度进行检测。
	该选项用于启动 `hw_server` 并在其器件表中添加额外的器件。默认情况下， `hw_server` 启动时显示预设列表，其中包含从 XICOM SQL jtag 主表编译输入的器件。可通过使用此 `set device-info-file` 选项来覆盖或扩展这些设置。指定的文件为 `.csv` 文件，该文件会忽略以“#”开头的行。
	在 `hw_server` 启动时指定 `.csv` 文件的方式如下：
	`hw_server -e "set device-info-file my_file.csv"`
	`.cvs` 文件格式如下 (`hw_server_device_info_file.csv`)：
	`### ### # File: hw_server_device_info_file.csv # Description: # This is a sample jtag ID table. This file can be used to define # additional devices to be detected by the hw_server application. # # In this file empty lines, lines with spaces, and lines that begin # with the '#' character will be ignored. # # The format of this file is as follows: # # ROW 1: fields # The standard JTAG id fields are "idcode,mask,irlen,name" # ROW 2: field types # For each field specified in ROW 1, the type is used to # interpret the field data read per device. The types # accepted are "i" for integer and "s" for string. If # you use the standard fields on ROW 1 then you should # use "i,i,i,s" in ROW 2 to set the fields idcode,mask # irlength to integer and name as string # # To use this table you start hw_server with the following # command line arguments: # # hw_server -e "set device-info-file <file>" # # Where <file> is replaced with this filename. ### ### idcode,mask,irlen,name i,i,i,s 167784595,268435455,10,chipscope_soft_tap`

`device-info-file`	设置要使用的默认器件文件 `.csv`

表 67：**高级选项** (续)

选项	用途
max-jtag-devices	增加在单一扫描链中可检测的器件的最大数量。默认值为 32。 该选项用于启动 hw_server 并且单一扫描链中可检测的数量多于默认器件数量。此设置的默认值为 32。您可增大该值以便延长 jtag 链。 **注释：**增大该数值会导致器件发现进程变慢，从而导致电缆接入速度变慢。因此，如果您要增大该值，此设置仅适用于含大量器件的系统。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set max-jtag-devices 64"`
xdb-user-bscan	此项可设置将用于扫描 xsdb 核的 bscan。 该选项用于启动 hw_server 用于在其他 bscan 上扫描 xsdb 主核。默认情况下，hw_server 会扫描用户 1 和 3 bscan。通过该选项，您可启动 hw_server 用于在其他 bscan 用户插槽中查找 bscan。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set xsdb-user-bscan 1,2,3,4"` 此参数的实参可用于指定参数列表。该列表是逗号分隔列表，范围为 1 至 4。列表指定的最小元素数量为 1，最大数量为 4。
mdm-detect-bscan-mask	此项可设置将用于扫描 mdm 核的 bscan。 该选项用于启动 hw_server 对其他 bscan 上 MicroBlaze 或 MicroBlaze V 主核的扫描。默认情况下，hw_server 将扫描用户 2 bscan。通过该选项，您可启动 hw_server，并将其用于在其他 bscan 用户插槽上查找 MicroBlaze 或 MicroBlaze V。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set mdm-detect-bscan-mask 2"` 位掩码适用于由 hw_server 发现的任意 FPGA 或自适应 SoC。以下是一些常见位掩码设置示例： 掩码值 BSCAN 扫描 0 无 1 User1 3 User1、User2 7 User1、User2、User3 f User1、User2、User3、User4

always-open-jtag	强制 hw_server 在启动时打开所有目标。
	默认情况下，当 hw_server 启动时，不会对任何电缆进行初始化。发起首个连接时，会发现并打开电缆。此发现周期耗时数秒至数分钟，因系统而异。完全初始化后，即可读取电缆并发现器件。
	在某些情况下，需要先发现电缆并备用。例如，将 Linux 系统设置为开发板服务器时，最好始终初始化电缆并准备好提供连接。对于此类情况，您可使用 always-open-jtag 选项来强制打开电缆。
	默认情况下，这是启动时使用的设置（无需传入任何实参）：
	`hw_server -e "set always-open-jtag 0"`
	要强制打开电缆，请传入以下实参，如下所示：
	`hw_server -e "set always-open-jtag 1"`

表 67：**高级选项** (续)

选项	用途

auto-open-servers	打开含用于打开 XVC 电缆的特定参数的电缆服务器。
	该选项为调试选项，用于自动打开 XVC 线缆连接，并控制 hw_server 应检测的 USB 线类型。auto-open-servers 参数的实参为由字段组成的逗号分隔列表。每个字段均为由子字段构成的冒号分隔列表，其中首个子字段即为其类型，而其余子字段则因类型而异。对于 xilinx-xvc 类型，子字段为 host 和 port。auto-open-servers 的默认值为“*”，表示 hw_server 应检测它可支持的所有电缆类型。需要参数的 Cables 类型（如 XVC）则不在“*”覆盖范围内。
	在 hw_server 启动时指定该选项的方式如下：
	`hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200"`
	要打开 2 台服务器，应使用：
	`hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200,xilinx-xvc:localhost:10210"`
	要打开 2 台 XVC 服务器，除了使用所有基于 USB 的电缆外，还应使用：
	`hw_server -e "set auto-open-servers *:xilinx-xvc:localhost:10200,xilinx-xvc:localhost:10210"`

auto-open-ports	控制端口（链或扫描链）的自动打开。 该选项用于控制 JTAG 扫描链的自动打开。auto-open-ports 参数的实参为布尔值，1 表示自动打开所有已知 JTAG 扫描链，0 表示客户端会打开选定的 JTAG 扫描链。默认值为 1。例如，当有多个 JTAG 扫描链连接到单一主机并使用 hw_server 的不同实例来访问每个扫描链时，即可将该值设置为 0。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set auto-open-ports 0"`
xvc-timeout	更改 XVC 超时值有助于调试 XVC 服务器。 该选项为调试选项，用于增大 XVC 传输事务终止所需的超时周期。xvc-timeout 参数的实参为以秒为单位的时间。值为 0 即禁用超时，这会导致无限等待。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set xvc-timeout 100"`

表 67：**高级选项** (续)

选项	用途

xvc-servers	启动 XVC 服务器以使用电缆。
	该选项用于使 hw_server 启动 XVC 服务器以使用指定 JTAG 线缆。xvc-servers 参数的实参为由 XVC 服务器描述组成的逗号 (,) 分隔列表。每条 XVC 服务器描述均为冒号 (:) 分隔列表，其中包含电缆标识、XVC 服务器主机名或编号以及端口号。电缆标识为制造商名称和唯一标识，以斜杠 (/) 字符分隔。电缆标识可以是完整电缆标识的一部分，并且主机名可为空。如果为空，则表示服务器应监听所有网络接口上的传入连接。
	请参阅 processor-debug-claim 以了解当 XVC 客户端与 XVC 服务器均为相同器件类型的调试器并使用 XVC 锁定模式时，如何避免干扰。
	在 hw_server 启动时指定该选项的方式如下：
	`hw_server -e "set xvc-servers 210203356596A:localhost:3000"`
	注释：您仍需首先连接到此 hw_server 实例，然后才能初始化电缆接口。
	如果不连接到电缆，那么您会看到如下消息：
	`TCF 19:11:02.417: XVC open port failed: Cannot find JTAG cable matching 210203A0314DA`
	要自动打开 XVC 线缆并将其锁定至 XVC，请添加 always-open-jtag 选项，如下所示：
	`hw_server -e 'set xvc-servers 210203A0314DA:xcoatslab-9:3122' -e 'set always-open-jtag 1'`

xvc-packet-len	更改 XVC 服务器的最大封装长度。 该选项用于控制使用 xvc-servers option 启动的 XVC 服务器所返回的 XVC 封装长度。当前的默认封装长度为 16000，但在后续更新版本中可以更改此设置。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-packet-len 1000"`
xvc-version	更改 XVC 服务器的 XVC 协议版本。 这是调试选项，可搭配 xvc-servers 一起使用，以控制对 XVC 客户端公开的 XVC 协议版本。当前的默认协议版本为 1.1，但如果定义了新版本的 XVC 协议，则可以更改此设置。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-version 1.0"`
xvc-capabilities	更改 XVC 服务器的 XVC 功能。 这是调试选项，可搭配 xvc-servers 一起使用，以控制对 XVC 客户端公开的功能。当前的默认功能包括 locking、status 和 state-aware，但在将来版本中可以添加其他功能。如果 xvc-version 设为 1.0，则该选项无效。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-capabilities status,state-aware"`

表 67：**高级选项** (续)

选项	用途
processor-debug-claim	自动申领所选器件类型，以阻止调试器使用这些类型。 该选项用于阻止 hw_server 调试器使用选定的器件类型进行调试。给予该选项的实参为位掩码。默认情况下，hw_server 使用所有已知的器件类型进行调试。位编号含义 位 器件类型 0 Arm DAP 1 MPSoC 2 FPGA 或自适应 SoC 在如下情况下可使用该选项：当使用 xvc-servers 启动 XVC 服务器并且 XVC 客户端作为先前一个或多个器件的调试器时，或者当 hw_server 连接到 XVC 服务器并且该服务器作为先前一个或多个器件的调试器时。在这两种情况下，仅当使用锁定功能时，此设置才有效，因为这样即可在调试器之间支持时间共享。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set processor-debug-claim 2"`
jtag-poll-delay	此项表示延迟值（以 uS 为单位）。默认值为 50000。 该选项属于轮询选项，用于降低 JTAG 轮询频率。JTAG 轮询频率为 2 次 JTAG 轮询操作之间的最短周期。默认值和最小值均为 50,000 uS。jtag-poll-delay 参数的实参是以 uS 为单位的数值。
help	显示 hw_server “e” 选项 该选项用于显示 hw_server 的所有可用 “e” 选项。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e help`
show-all	显示 hw_serv 中设置的所有成功的选项。 该选项用于显示 hw_server 的所有 “e” 选项设置。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e show-all`
jtag-default-frequency	设置所有电缆的默认频率。 该选项用于设置 JTAG TCK 默认频率。jtag-default-frequency 参数是以 Hz 为单位的数值。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set jtag-default-frequency 5000000"`

表 67：**高级选项** (续)

选项	用途

jtag-port-filter	设置 JTAG 端口筛选。
	该选项用于控制 JTAG 端口筛选。设置该选项后，hw_server 会忽略与筛选不匹配的所有 JTAG 端口。此参数的实参为完整或部分端口标识组成的逗号分隔列表。端口标识是 <manufacturer>/<productid>/<serial><port> 形式的字符串。当在端口标识字符串中可找到任意端口筛选工具的字符串时，即表示筛选结果匹配。
	当在同一个主机上运行 hw_server 的多个实例时，可使用此参数来指定应由 hw_server 处理的电缆。
	在 hw_server 启动时指定该选项的方式如下：
	`hw_server -e "set jtag-port-filter Xilinx/DLC10/0000128f515601"`
	另一个示例是在端口 3122 上启动 hw_server 时筛选任意 AMD DLC9 或 DLC10 电缆：
	`hw_server -stcp::3122 -e "set jtag-port-filter DLC9,DLC10"`

bscan-switch-user-mask	启用 bscan 切换。 该选项用于控制 bscan 切换检测。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set bscan-switch-user-mask <user-bit-mask>"`
jtag-port-devices	设置 JTAG 端口器件列表。 该选项用于指定 JTAG 扫描链的器件静态列表。指定该选项后，hw_server 不会读取 IDCODE 寄存器以检测扫描链上的器件。当扫描链包含不遵循 IEEE 1149.1 规范的器件时，适用该选项。给予该参数的值为 IDCODE 值组成的逗号分隔列表，其中的值的顺序与扫描链上的器件顺序相同。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set jtag-port-devices 0xe970203f,0x03632093"`
max-ir-length	启用 JTAG 链中的器件，这些器件的 ir 长度大于 64 位。 该选项用于启动 hw_server 并且能够启用大于 64 位的 ir 长度。此设置的默认值为 64。对于 JTAG 链中 ir 长度较宽（例如，93）的器件，可以增大该值。 **注释：**增大该数值会导致器件发现进程变慢，从而导致电缆接入速度变慢。 因此，如果您要增大该值，此设置仅适用于含大量器件且 ir 长度较长的系统。 在 hw_server 启动时指定该选项的方式如下： `hw_server -e "set max-ir-length 93"`

附加资源与法律声明

查找其他文档

技术信息门户网站

AMD 技术信息门户网站是旨在使用您的网页浏览器提供健全的文档搜索和导航的在线工具。要访问该技术信息门户网站，请转至 https://docs.amd.com。

注释：单击链接将打开英语版本，但您可从下拉列表中选择简体中文版本（如可用）。请注意，简体中文版本可能比英语版本旧。

Documentation Navigator

Documentation Navigator (DocNav) 是预安装的工具，支持访问 AMD 自适应计算文档、视频和支持资源，您可在其中通过筛选和搜索来查找信息。要打开 DocNav，请执行以下操作：

- 在 AMD Vivado™ IDE 中，单击“Help” → “Documentation and Tutorials”。
- 在 Windows 上，单击“Start”（开始）按钮并选中“Xilinx Design Tools” → “DocNav”。
- 在 Linux 命令提示中输入 `docnav`。

注释：如需了解有关 DocNav 的更多信息，请参阅《Documentation Navigator 用户指南》(UG968)。

注释：您无法从 DocNav 访问简体中文版本。请使用设计中心网页。

设计中心

AMD 设计中心提供了根据设计任务和其他主题整理的文档链接，可供您用于了解关键概念以及常见问题解答。要访问设计中心，请执行以下操作：

- 在 DocNav 中，单击“Design Hubs View”选项卡。
- 转至设计中心网页。

支持资源

如需获取答复记录、技术文档、下载以及论坛等支持资源，请访问技术支持。

培训资料

AMD 提供多种多样的培训课程和 QuickTake 视频，可帮助您进一步了解本文档中提出的概念。请使用以下链接获取相关培训：

1. 使用 Vivado Design Suite 设计 FPGA 1 (FPGA-VDES1)
2. 使用 Vivado Design Suite 设计 FPGA 2 (FPGA-VDES2)
3. 使用 Vivado Design Suite 设计 FPGA 3 (FPGA-VDES3)
4. 使用 Vivado Design Suite 设计 FPGA 4 (FPGA-VDES4)
5. Vivado Design Suite QuickTake 视频：使用 Vivado IP integrator 定位 Zynq 器件
6. Vivado Design Suite QuickTake 视频：在 Vivado Design Suite 中进行部分重配置
7. Vivado Design Suite QuickTake 视频：使用 Vivado Design Suite 版本控制
8. Vivado Design Suite QuickTake 视频教程
9. 嵌入式设计教程：面向含 SmartLynq+ 模块的高速调试端口的系统设计示例

参考资料

以下技术文档是非常实用的补充资料，可配合本指南一起使用：

1. Vivado Design Suite 文档
2. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
3. 《Vivado Design Suite 用户指南：综合》(UG901)
4. 《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)
5. 《Vivado Design Suite 教程：Dynamic Function eXchange》(UG947)
6. 《适用于 FPGA 和 SoC 的 UltraFast 设计方法指南》(UG949)
7. 《Vivado Design Suite 用户指南：实现》(UG904)
8. 《Vivado Design Suite 用户指南：版本说明、安装和许可》(UG973)
9. 《Vivado Design Suite 用户指南：设计流程概述》(UG892)
10. 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899)
11. 《Vivado Design Suite 教程：烧录和调试》(UG936)
12. 《Vivado Design Suite Tcl 命令参考指南》(UG835)
13. 《SmartLynq 数据线缆用户指南》(UG1258)
14. 《7 系列 FPGA 配置用户指南》(UG470)
15. 《7 系列 FPGA 与 Zynq 7000 SoC XADC 双 12 位 1 MSPS 模数转换器用户指南》(UG480)
16. 《UltraScale 架构配置用户指南》(UG570)
17. 《UltraScale 架构系统监控器用户指南》(UG580)

18. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)
19. 《UltraScale + FPGA GTM 收发器用户指南》(UG581)
20. 《Debug Bridge LogiCORE IP 产品指南》(PG245)
21. 《在 Zynq 7000 上使用 PetaLinux 工具运行 AMD 虚拟线缆》(XAPP1251)
22. 《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940)
23. 《使用嵌入式微控制器（ISE 工具）进行赛灵思系统内编程》(XAPP058)
24. 《使用加密确保 7 系列 FPGA 比特流的安全》(XAPP1239)
25. 《使用加密和身份验证确保 UltraScale/UltraScale+ FPGA 比特流的安全》(XAPP1267)
26. 《Virtual Input/Output LogiCORE IP 产品指南》(PG159)
27. 《Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP 产品指南》(PG132)
28. 《Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP 产品指南》(PG133)
29. 《Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP 产品指南》(PG152)
30. 《Integrated Logic Analyzer LogiCORE IP 产品指南》(PG172)
31. 《JTAG to AXI Master LogiCORE IP 产品指南》(PG174)
32. 《System Integrated Logic Analyzer LogiCORE IP 产品指南》(PG261)
33. 《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)
34. 《Debug Bridge LogiCORE IP 产品指南》(PG245)
35. 《In-System IBERT LogiCORE IP 产品指南》(PG246)
36. 《UltraScale 架构 FPGA 存储器 IP LogiCORE IP 产品指南》(PG150)
37. 《AXI High Bandwidth Controller LogiCORE IP 产品指南》(PG276)
38. 《IBERT for UltraScale GTM Transceivers LogiCORE IP 产品指南》(PG342)
39. 《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)
40. 《Bootgen 用户指南》(UG1283)
41. 《适用于 FPGA 和 SoC 的 UltraFast 设计方法指南》(UG949)
42. 《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)
43. 《Vivado Design Suite 教程：Dynamic Function eXchange》(UG947)
44. 《Versal Adaptive SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)
45. 《Vivado Design Suite Tcl 命令参考指南》(UG835)
46. 《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)
47. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)

修订历史

下表列出了本文档的修订历史。

章节	修订综述
2024 年 11 月 13 日 2024.2 版	
第 5 章：利用 pdi_dbg_util 调试 PDI 烧录	添加章节。
附录 F: 配置存储器支持	更新闪存器件表。
第 17 章：Versal 串行 I/O 硬件调试流程	添加注释。
2024 年 5 月 30 日 2024.1 版	
Trigger At Startup	添加注释。
附录 F: 配置存储器支持	更新闪存器件表。

请阅读：重要法律声明

本文档所示信息仅做参考，其中可能包含不准确的技术信息、疏漏和印刷错误。受诸多原因影响，此处所含信息可能发生更改，也可能无法准确呈现，这些原因包括但不限于产品和路线图变更、组件和主板版本更改、新增模型和/或产品发布、不同制造商之间存在的产品差异、软件更改、BIOS 刷新、固件升级等。任何计算机系统均存在安全性漏洞风险，无法彻底阻止也无法缓解这类风险。AMD 没有任何义务来更新或者以任何其他方式纠正或修改这些信息。但 AMD 保留随时修改这些信息和更改文档内容的权利，AMD 没有任何义务将此类修改或更改通知任何人。此处信息“按原样”提供。AMD 对于本文档内容不作任何陈述或保证，并且对于这些信息中可能出现的任何不准确、错误或疏漏问题不承担任何责任。对于有关任何暗含的非侵权、适销性及适合特定用途的保证，AMD 特此声明不承担任何责任。无论在任何情况下，对于任何人因使用此处包含的任何信息而形成的依赖或者引发的任何直接、间接、特殊或其他后果性损害，AMD 概不负责，即使 AMD 已明确获悉存在发生此类损害的可能性也是如此。

关于与汽车相关用途的免责声明

如将汽车产品（部件编号中含“XA”字样）用于部署安全气囊或用于影响车辆控制的应用（“安全应用”），除非有符合 ISO 26262 汽车安全标准的安全概念或冗余特性（“安全设计”），否则不在质保范围内。客户应在使用或分销任何包含产品的系统之前为了安全的目的全面地测试此类系统。在未采用安全设计的条件下将产品用于安全应用的所有风险，由客户自行承担，并且仅在适用的法律法规对产品责任另有规定的情况下，适用该等法律法规的规定。

版权声明