LabVIEW FPGA 模块用户手册

2025-03-26

目录

欢迎使用LabVIEW FPGA模块用户手册

LabVIEW FPGA模块用户手册详细介绍了产品功能和使用步骤。

还想了解其他信息？

对于产品用户手册中未包含的信息（如规格或API参考），请浏览“相关信息”部分。

相关信息：

- 下载LabVIEW FPGA模块
- 许可证设置和激活
- LabVIEW FPGA模块发行说明
- LabVIEW高性能FPGA开发者指南
- LabVIEW用户手册
- CompactRIO开发者的LabVIEW指南
- NI培训中心

FPGA模块是什么?

FPGA模块是一款LabVIEW的附加软件，可提供高度集成的开发环境、IP库、高保真仿真器和调试功能，用户可以更高效地设计基于FPGA的系统。使用FPGA模块和LabVIEW创建在NI FPGA终端上运行的VI。可重配置I/O (RIO)设备可用作FPGA终端。FPGA终端包含一个由固定大小的I/O资源包围的可重配置FPGA（现场可编程门阵列）。根据具体的FPGA终端，固定的I/O资源可包括模拟和数字资源－例如，模数转换器(ADC)和数模转换器(DAC)，通过FPGA可控制上述资源。

通过FPGA模块配置可重配置FPGA的动作以满足具体测量和控制系统的要求。运行于FPGA终端的VI称为***FPGA VI***。FPGA模块可用于创建FPGA VI。下载FPGA VI至FPGA的过程是对FPGA终端执行编程操作。用户创建和下载的每个新建FPGA VI均为一个自定义定时、触发和I/O解决方案。

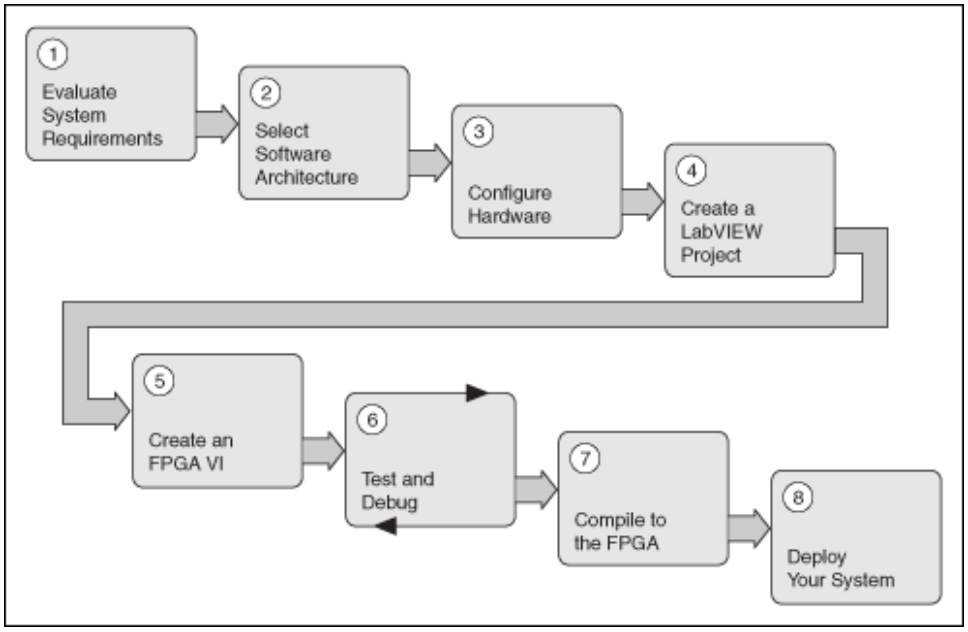
浏览相关概念，了解可帮助您开始构建LabVIEW FPGA应用程序的主题。

相关概念：

- FPGA应用和项目简介
- FPGA编程概述
- FPGA硬件概述
- FPGA终端硬件文档

建立设计流程

根据FPGA应用程序的复杂程度，用户可能希望快速编写和编译程序，或者希望利用内置仿真器调试、测试和验证更改过的代码，而无需每次都执行编译。下图为推荐的FPGA设计流程图。它以评估系统要求开始，以部署系统结束。



安装LabVIEW FPGA模块

可以使用NI Package Manager或LabVIEW开发平台安装盘安装LabVIEW FPGA模块。

应用软件支持

LabVIEW 2024 Q1 FPGA模块支持LabVIEW 2024 Q1。关于LabVIEW FPGA模块早期版本的应用软件支持信息，请参阅这些版本的自述文件。

使用Package Manager安装LabVIEW FPGA模块

关于使用Package Manager安装、移除、更新NI软件的信息，请参见***Package Manager***。

使用LabVIEW开发平台安装盘安装LabVIEW FPGA模块

1. 以管理员身份或具有管理员权限的用户登录。
2. 插入LabVIEW开发平台安装盘。如需获取额外的LabVIEW开发平台安装盘，请访问NI网站。如您购买的是NI软件套件或产品组合，请使用随附的安装光盘安装产品。
3. 按照屏幕上出现的提示依次安装和激活下列软件：

- LabVIEW
- FPGA模块
- （可选）LabVIEW Real-Time模块 – 使用LabVIEW Real-Time模块编程NI PXI、NI PXI Express、NI CompactRIO和NI单板RIO设备上的实时操作系统。产品包含一个临时许可证。
- Xilinx编译工具 – 可在开发计算机上安装这些编译工具，以本地编译LabVIEW FPGA VI。如要使用配置组件级IP向导、配置IP集成节点、整合Xilinx IP或第三方仿真，必须在开发计算机上安装编译工具。关于设备中FPGA芯片的详细信息，见相应的硬件文档并安装相应的工具集。关于每种Xilinx编译工具支持的NI硬件的详细信息，请访问***使用信息代码***并输入信息代码XilinxCompileTools查询。
  - Xilinx编译工具(Windows) – 在使用Windows操作系统的计算机上编译FPGA VI时，安装这些编译工具。Windows上推荐的编译工具是Vivado 2021.1。Windows 10不支持ISE编译工具。

    注： 如要配置IP集成节点或针对Virtex-II设备使用配置组件级IP向导，也必须安装用于ISE 14.7的Xilinx编译工具。Virtex-II FPGA设备不支持Xilinx IP和仿真导出。

  - （可选）FPGA Compile Farm Server – 使用FPGA Compile Farm Server在多个远程计算机间分发FPGA VI编译任务。

    注： 使用LabVIEW FPGA Compile Cloud Service，通过云环境无需将编译任务加载至多个计算机，从而提高开发效率。

  - 设备驱动程序 – 包含大多数FPGA终端的驱动程序软件。关于用户所需其他或不同的设备驱动程序的详细信息，见相应的FPGA终端硬件文档。

安装程序除了将程序文件和文档安装在LabVIEW目录下之外，还可从Xilinx中提取文件放在x:\NIFPGA目录下，其中x为安装LabVIEW的磁盘。FPGA模块使用这些文件将FPGA VI编译为可在FPGA终端运行的代码。

设置远程编译场服务器或编译工作站

可以使用LabVIEW开发平台安装包安装FPGA Compile Farm Server，取消加载编译至

指定计算机。必须在用于管理编译的远程计算机上，独立安装FPGA Compile Farm Server，且必须在用作远程编译工作站的计算机上安装Xilinx编译工具。关于每种Xilinx编译工具支持的NI硬件的详细信息，请访问***使用信息代码***并输入信息代码XilinxCompileTools查询。

- Windows – 关于在远程计算机上设置FPGA Compile Farm Server的信息，请访问***使用信息代码***并输入信息代码FPGAkb1rcs查询。
- Linux – 关于设置远程编译工作站的信息，见***FPGA Module Xilinx Compilation Tools for Linux Readme***。

NI提供了***LabVIEW FPGA Compile Cloud Service***，通过云环境无需将编译任务加载至多个计算机，从而提高开发效率。关于LabVIEW FPGA编译云的信息，请访问***NI LabVIEW FPGA Compile Cloud Service***。

相关信息：

- Package Manager
- 使用信息代码
- NI LabVIEW FPGA编译云服务
- FPGA Module Xilinx Compilation Tools for Linux Readme

新增功能和改动

了解LabVIEW FPGA模块各个版本的新增功能和行为变化等更新。

LabVIEW FPGA模块2022年第3季度改动

- 添加了新的I/O模块属性节点。使用此节点和**模块**常量可查看CompactRIO设备的属性。

LabVIEW FPGA模块2020年改动

- 支持C系列FPGA终端。
- 打开FPGA VI引用节点具有下列更改：
  - **转换为定点**选项卡已停用。该选项曾提供了将浮点数据类型转换为定点的工具。
  - RIO地址输入重命名为设备名称。
  - **从文件部署**运行模式不再可用。
- 运行FPGA仿真节点重命名为运行GCDL仿真。
- 设置输出启用(CDL)节点有以下改动：
  - 添加了错误输入和错误输出接线端。
  - 启用输入端重命名为启用？。

LabVIEW FPGA模块2019年改动

- 访问CDL循环外的存储器项。
- 移除了I/O通道节点不起作用的接线端。
- 执行下列任务时不再需要LabVIEW FPGA模块：
  - 在SystemDesigner的“设计”视图选板中访问FPGA终端。
  - 查看或编辑FPGA代码。
  - 部署比特文件(.lvbitx)至FPGA终端。
  - 迁移LabVIEW FPGA函数至LabVIEW。

- 执行下列任务时需要LabVIEW FPGA模块及相关驱动程序：
  - 在FPGA代码中使用驱动程序特有功能。
  - 运行或编译FPGA代码。
  - 将依赖特定驱动程序的LabVIEW FPGA代码迁移至LabVIEW。
- 在本地编译FPGA代码，无需手动设置FPGA编译器环境。
- 配置FPGA资源常量，以在资源名称中显示命名空间。

FPGA终端硬件文档

LabVIEW文档包含几个FPGA终端硬件文档的引用，提供了额外的终端特定信息。每个FPGA终端都附带额外的文档，对开发FPGA VI可能大有裨益。***LabVIEW文档***包含了大多数FPGA终端的主题。此外，某些FPGA终端在其它路径也带有帮助文档。如要定位其它文档资源见每个终端的相关文档主题。

有时，有些功能为FPGA设备系列（如Virtex-II或Virtex-5）特有。有关FPGA功能的信息，见Xilinx文档。

相关概念：

- FPGA模块是什么？
- 创建至点对点读取方FIFO的引用
- 创建点对点写入方FIFO的引用
- 创建点对点读取方FIFO
- 创建点对点写入方FIFO
- 配合使用点对点数据流和FPGA终端

FPGA编程概述

对于需要实现下列功能的可编程应用，LabVIEW FPGA模块为最佳选择：

- **自定义I/O** – 可使用自定义计数器、编码器和脉宽调制器(PWM)修改数字和模拟线。
- **板载决策** – 可在终端执行控制、数字滤波和布尔判定。
- **资源同步** – 应用程序可以FPGA终端资源的精确定时运行。这些资源可为模拟输入(AI)、模拟输出(AO)、数字输入和输出(DIO)、计数器及PWM。应用程序可在多个FPGA终端间被同步。
- **并行执行** – 程序框图的独立部分可在FPGA中并行执行。例如，程序框图上的多个独立While循环每个都可在FPGA的独立部分同步运行。添加额外的独立循环不会影响现有循环的性能。
- **独立和确定性的执行** – 即使控制和监视FPGA终端的计算机崩溃，FPGA VI也可继续运行。

使用LabVIEW编程FPGA

下文为编程FPGA终端的步骤概述。该列表不能用作完整的使用说明。***LabVIEW文档***中包含各式主题，有助于了解开发FPGA应用程序的详情。

1. **了解终端的硬件性能** – 关于FPGA和终端的性能及功能的信息，见特定的FPGA终端或机箱硬件文档。
2. **创建应用的FPGA项目** – 开发FPGA应用程序前，必须创建一个带有FPGA终端的LabVIEW项目。
3. **创建FPGA VI** – 可新建或通过范例VI创建FPGA VI。使用NI范例查找器查找适用于终端的范例VI。
4. **（可选）创建主控VI** – 主控VI在RT终端或PC上运行，可控制和监视FPGA VI。
5. **编译和下载FPGA VI至终端** – 必须先编译FPGA VI才能将其下载至FPGA终端并在终端运行。

相关概念：

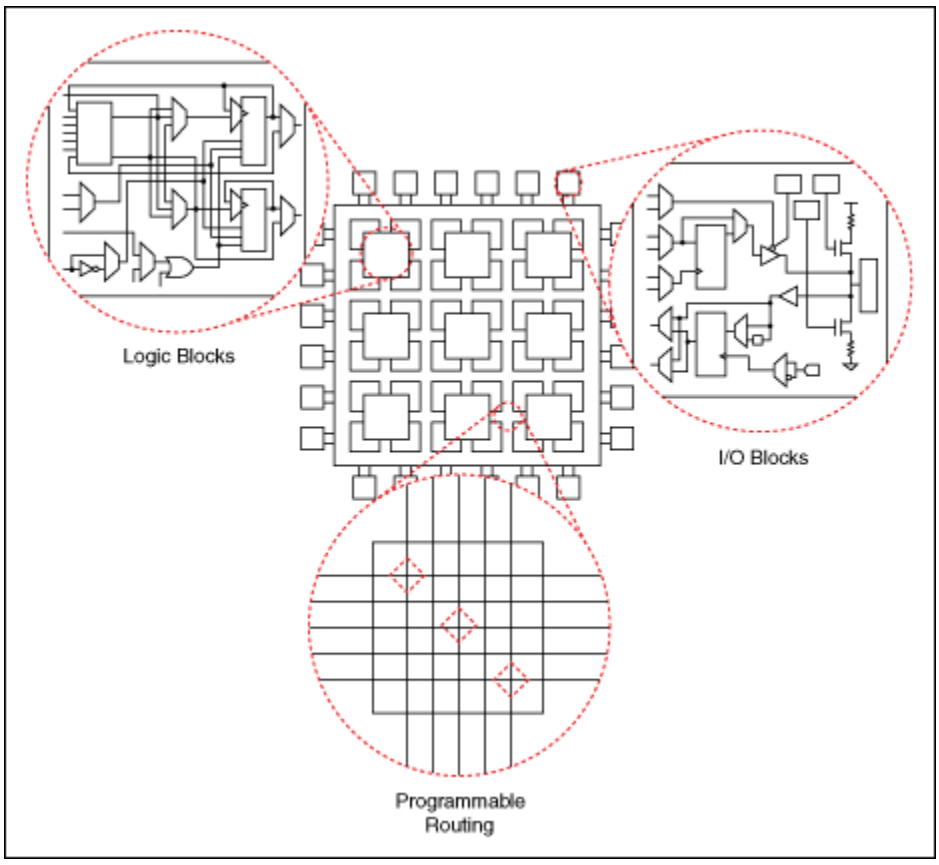
- FPGA模块是什么?
- FPGA应用和项目简介
- 通过主控计算机与FPGA终端通信

FPGA硬件概述

每个FPGA芯片(FPGA)是由有限个带有可编程连接预定义资源组成。这些互连资源通过LabVIEW FPGA模块实现用户设计的数字电路。用户创建FPGA VI时设计一个电路示意图，以说明FPGA逻辑块的连接方式。编译VI时，编译工具将FPGA VI转译为FPGA电路。

注： 本文将介绍与底层FPGA实现相关的概念。理解这些概念并不是使用LabVIEW FPGA模块的前提，但是本文有助于提高编写FPGA VI的效率。

下图显示了FPGA上逻辑块、I/O块和可编程线路之间的关系。



设计FPGA应用时，下列FPGA规范为重要的考虑因素。

- 可配置逻辑块的数量
- 固定函数逻辑块的数量（例如，乘法器）
- 存储器资源的大小（例如，嵌入式块RAM）

关于FPGA基础的详细信息，请参考ni.com网站上的的技术支持文档。

触发器、LUT和逻辑片

FPGA资源为FPGA上可执行逻辑功能的资源。FPGA资源被组合在逻辑片内，以创建可配置逻辑块。每个逻辑片包含一组LUT、触发器和多路复用器。LUT是FPGA上一组硬接线的逻辑门。LUT是存储每个输入组合的预定义输出列表的逻辑块，因此能够快速获取逻辑操作的输出。触发器是能够实现两个稳态，表示一个比特的电路。多路复用器（通常称为MUX）在两个或多个输入之间选择输入，并输出选中的输入。

不同的FPGA系列执行逻辑片和LUT的方式不同。例如，Virtex-II FPGA上的逻辑片带有两个LUT和两个触发器。Virtex-5 FPGA上的逻辑片带有四个LUT和四个触发器。此外，LUT的输入端数量通常为2至6个，实际数量取决于FPGA的型号。

寄存器

寄存器是存储位模式的一组触发器。FPGA上的寄存器带有时钟、输入数据、输出数据和启用信号端口。在每个时钟周期内，输入数据被触发且存储在内部，然后更新输出数据为与内部存储数据匹配的值。FPGA VI使用寄存器执行下列功能：

- 在循环计数间保持状态
- I/O同步
- 时钟域间的握手数据
- 流水线
- 与主机VI通信

寄存器是了解FPGA VI的定时考虑因素的重要概念。

块RAM

块存储器（块RAM）是嵌入在FPGA上用于存储数据的随机存储器。通常，LabVIEW使用块存储器实现存储器和FIFO函数。用户可指定LabVIEW分别使用FIFO属性和存储器属性对话框实现FIFO和存储器项。

DSP48E和DSP48E1

DSP48E逻辑片是特定FPGA系列（例如，Xilinx Virtex-5）包含的数字信号处理逻辑单元。该逻辑片可用于实现不同的算术运算。例如，乘法累加器、乘加器、单步/n步计数器。该逻辑片还可用于执行AND、OR、XOR等逻辑运算。使用多个DSP48E逻辑片可在不占用额外FPGA架构资源的前提下实现更为复杂的功能。例如，实现复杂乘法器或n阶FIR滤波器。

DSP48E1是某些Xilinx Virtex-6或更高级别FPGA设备的数字信号处理单元。DSP48E1在多个方面扩展了DSP48E逻辑片的功能（例如，在乘法器前包含一个预加器）。该预加器对执行特定的算法非常有用（例如，对称FIR滤波器）。

UltraRAM

UltraRAM (URAM)是一种高度灵活的低密度大容量存储块，可用于大多数Xilinx UltraScale+终端。UltraRAM的容量是块存储器的8倍，但数据宽度和地址空间配置的灵活性不如块存储器。UltraRAM可用于存储较大的本地FIFO。在FIFO属性对话框中可以指定LabVIEW如何实现FIFO。

相关概念：

- FPGA模块是什么？
- 配置DSP48E和DSP48E1函数
- FPGA VI的数据流和启用链
- 使用流水线优化FPGA VI
- FPGA VI的定时考虑因素
- 限制FPGA VI中顶层前面板对象的数量
- 在项目浏览器窗口管理FPGA应用

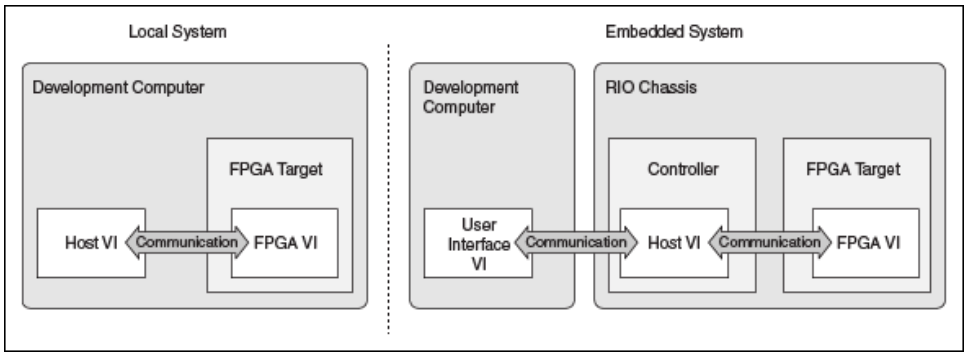
- 优化FPGA VI的执行速度和大小

FPGA应用和项目简介

FPGA应用可以是带FPGA终端并在开发计算机上运行单个FPGA VI的本地系统；或是包含多个FPGA终端、一个或多个RT终端和运行在开发计算机上的LabVIEW的大型嵌入式系统。必须使用LabVIEW项目文件(`.lvproj`)创建FPGA应用程序。理解系统架构能够帮助用户使用LabVIEW项目创建和管理FPGA应用。

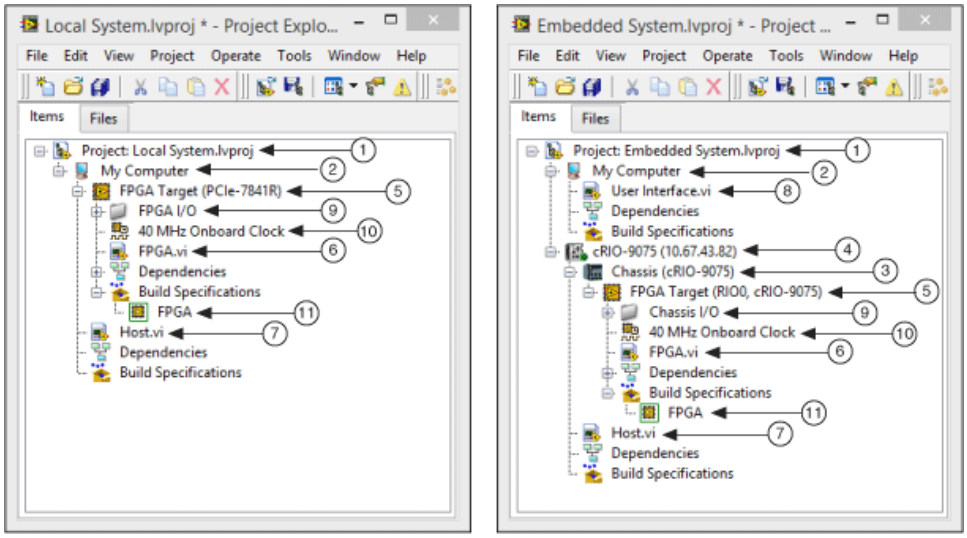
理解本地和嵌入式系统

LabVIEW项目结构取决于FPGA应用中包含的组件。参见下图理解包含FPGA终端的本地系统和嵌入式系统的物理布局。



FPGA项目组成部分

使用**项目浏览器**窗口管理FPGA项目。每个FPGA项目随系统架构的组成部分改变。下图展示了本地系统`Local System.lvproj`和嵌入式系统`Embedded System.lvproj`的组成部分。



下表介绍了FPGA应用组成部分及其在LabVIEW FPGA项目中的位置。

	组件	说明
①	LabVIEW项目	LabVIEW项目允许用户在开发计算机上开发应用时，管理VI和终端。LabVIEW项目文件包含对项目中文件的引用、配置信息、部署信息、生成信息等。
②	我的电脑	我的电脑（通常称为开发计算机）是开发LabVIEW项目的计算机。开发计算机是指运行支持的Windows平台的计算机，该计算机已安装LabVIEW和LabVIEW FPGA模块。
③	RIO机箱	RIO机箱放置并直接连接FPGA终端的I/O块至可互换的I/O模块，以实现高性能定时、触发和同步。
④	控制器	控制器直接连接RIO机箱，直接或通过网络与开发计算机通信。控制器中包含一个嵌入式处理器，可运行实时操作系统(RTOS)或Windows。
⑤	FPGA终端	FPGA终端是一个可编程芯片，由逻辑块、I/O块和可编程互连资源组成，这些资源用于实现通过LabVIEW FPGA模块设计的数字电路。
⑥	FPGA VI	FPGA VI是加载至FPGA终端并运行的VI。LabVIEW编译工具将FPGA VI转换为电路机制，以重新配置FPGA终端块和互连资源。
⑦	主控VI	主控VI在控制器上运行，通过编程与FPGA VI通信。可使用主机VI记录数据、控制数据传输的定时及创建将FPGA终端作为组件的系统。

	组件	说明
⑧	用户界面VI	用户界面VI在开发计算机上运行，并与主控VI通信。用户界面VI允许用户通过编程，与主控VI的输入控件和显示控件交互。在没有控制器的情况下，用户界面VI将成为主机VI并直接与FPGA VI进行通信。
⑨	I/O	I/O是指FPGA系统的模拟和数字输入/输出。例如，热电偶、RTD、桥传感器、计数器和发生器等。关于支持的I/O的详细信息见指定硬件文档。
⑩	时钟	时钟通过指定FPGA系统的定时需求，控制FPGA VI的执行频率。如未包含额外的用于控制定时的代码，操作的执行频率由VI数据流确定。多数FPGA终端的默认时钟速率为40 MHz。
⑪	程序生成规范	FPGA终端的程序生成规范指定用户编译FPGA VI时，LabVIEW创建哪个选项： • **仿真导出** – 配置和导出用于第三方仿真的项目文件。 • **编译** – 配置和转换待下载至FPGA终端的项目文件为比特文件。 • **源代码发布** – 配置待发布的项目文件为独立应用程序。 编译、下载和运行FPGA VI前，必须创建程序生成规范。如未创建程序生成规范，LabVIEW将自动创建和指定默认的程序生成规范。

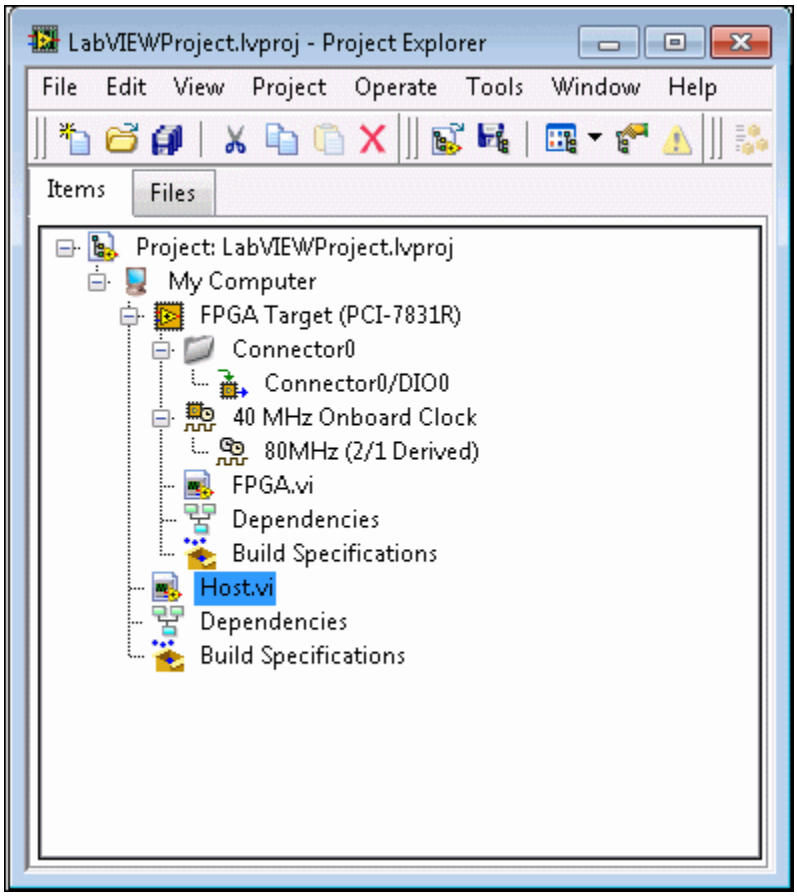
相关概念：

- FPGA模块是什么?
- FPGA编程概述
- 在项目浏览器窗口管理FPGA应用
- 添加FPGA终端至LabVIEW项目
- 添加项至项目浏览器窗口中的FPGA终端
- 创建FPGA VI
- 编译、下载和运行FPGA VI

在项目浏览器窗口管理FPGA应用

使用项目浏览器窗口管理FPGA应用的组件，包括FPGA VI和主控VI、FPGA终端和终端特定选项（例如，FPGA I/O、FPGA FIFO和FPGA终端时钟）。下图显示了包含FPGA终端、FPGA时基时钟和衍生时钟、FPGA VI、FPGA I/O项、FIFO和主控VI的**项**

目浏览器窗口。



必须为FPGA VI和主控VI创建项目。可以使用FPGA项目向导创建项目。也可以在“启动向导”窗口单击**新建项目**或在LabVIEW中选择**文件»新建**，在“新建”对话框中选择**项目»空项目**，创建新项目。现在，可将FPGA终端、FPGA终端时钟、FPGA I/O和FPGA FIFO添加至项目。FPGA终端可选择之前在NI Measurement & Automation Explorer (MAX)中配置的或添加一个新的FPGA终端。在项目浏览器窗口可添加、配置和管理FPGA终端下的VI、文件夹及FPGA项目项。

如要添加项至FPGA终端，在项目浏览器窗口右键单击终端，从快捷菜单中选择**新建»x**，其中x是指待添加项的类型，例如VI、FPGA I/O项或FIFO。添加的项位于项目浏览器窗口的FPGA终端目录树下。

使用基于终端的功能

创建FPGA VI与创建运行在Windows平台的VI类似。但有效编程FPGA需要了解终端的

功能以及基本的FPGA硬件概念。

项目浏览器窗口中FPGA终端下的每个项均包含FPGA终端特定信息和功能。在**项目浏览器**窗口中选择FPGA终端下的项时，LabVIEW仅显示FPGA终端支持的选项。例如，如在**项目浏览器**窗口选择FPGA终端下的FPGA VI并查看程序框图，LabVIEW会仅显示FPGA终端支持的函数选板的子选板、VI和函数。

> **注：** 如项目浏览器窗口中的FPGA终端带有感叹号图标⚠，则计算机上未安装终端支持。用户可查看和复制项目中的项，但只有安装终端支持后才能编译和运行项目。关于所需驱动程序、安装和配置FPGA终端的更多信息，见相关的硬件说明文档。

管理多个FPGA终端

在项目浏览器窗口可重用不同FPGA终端间的项。但新的FPGA终端未必支持前一个FPGA终端的功能，因此必须更新项属性，以匹配新的FPGA终端的功能和资源。例如，在2个不同的FPGA终端间复制FPGA I/O项时，必须验证目标FPGA终端是否支持源FPGA终端分配给FPGA I/O项的I/O资源。

相关概念：

- FPGA硬件概述
- 重用FPGA对象
- FPGA应用和项目简介

添加FPGA终端至LabVIEW项目

使用FPGA终端创建应用程序前，必须创建一个LabVIEW项目。然后添加FPGA终端至该项目并创建FPGA VI。

通过下列步骤之一添加FPGA终端至项目：

- 在本地用于开发的计算机上添加一个FPGA终端

- 在Real-Time系统中添加一个FPGA终端
- 在远程计算机上添加FPGA终端

创建带有FPGA终端的项目后可执行下列任务：

- 创建FPGA VI
- 添加FPGA I/O
- 添加FPGA时钟
- 添加FPGA寄存器项
- 添加FPGA存储器项
- 添加FPGA FIFOs

FPGA VI功能随具体FPGA终端变化。详细信息见具体FPGA终端的硬件文档有关FPGA终端功能的说明。

在本地用于开发的计算机上添加一个FPGA终端

按照下列步骤添加一个安装在本地用于开发的计算机上的FPGA终端至项目浏览器窗口。

1. 新建项目或打开现有项目。
2. 右键单击**项目浏览器**窗口中的**我的电脑**，从快捷菜单中选择**新建»终端和设备**，打开添加终端和设备对话框。
3. 选择**现有终端或设备**显示安装在用于开发的计算机上的FPGA终端。

   提示 如无FPGA终端，可选择**新终端或设备**，以显示用于开发的计算机上支持的FPGA终端类型。上述终端可被添加至项目以替代实际的终端。

4. 展开**FPGA终端**文件夹。LabVIEW显示安装在本地计算机上的FPGA终端和安装在本地计算机的终端的支持。
5. 从可用的终端列表中选择一个FPGA终端
6. 单击**OK**按钮。FPGA终端位于**项目浏览器**窗口的**我的电脑**下。

当前可在FPGA终端下添加一个FPGA VI，并开发该FPGA VI。

在Real-Time系统中添加一个FPGA终端

按照下列步骤添加RT系统中的FPGA终端至RT控制器下的**项目浏览器**窗口。某些FPGA终端不能用于RT控制器下。

1. 使用Real-Time新建项目向导创建项目和RT终端。
2. 右键单击**项目浏览器**窗口中的RT终端，从快捷菜单中选择**新建»终端和设备**显示添加终端和设备对话框。
3. 选择**现有终端或设备**显示安装在RT系统上的FPGA终端。

> **注：** 可能需要设置RT控制器的访问权限，以在**添加终端和设备**对话框中显示现有的FPGA终端。

4. 从可用的终端列表中选择一个FPGA终端
5. 单击**确定**。FPGA终端位于**项目浏览器**窗口的RT终端下。

当前可在FPGA终端下添加一个FPGA VI，并开发该FPGA VI。

在远程计算机上添加FPGA终端

MAX显示远程系统后，可设置其访问权限，还可在LabVIEW的**项目浏览器**中添加该系统，开发在该终端上运行的VI。按照下列步骤添加远程系统和其FPGA终端至LabVIEW的**项目浏览器**窗口。

1. 右键单击**项目浏览器**窗口中项目根目录，从快捷菜单中选择**新建»终端和设备**显示添加终端和设备对话框。
2. 选择**新终端或设备**并从列表中选择**联网计算机/设备**，然后单击**确定**。**项目浏览器**窗口中将显示该远程系统。
3. 在**项目浏览器**窗口中右键单击**联网计算机/设备**，从快捷菜单中选择**属性**显示联网计算机/设备属性 对话框。
4. 在**地址**文本框中为新添加的远程计算机分配一个IP地址。
5. 单击**OK**按钮。

6. 右键单击新添加的远程系统，从快捷菜单中选择**新建»终端和设备**。
7. 选择**现有终端或设备**显示安装在用于开发的计算机上的FPGA终端。

> 如无FPGA终端，可选择**新终端或设备**，以显示用于开发的计算机上支持的FPGA终端类型。上述终端可被添加至项目以替代实际的终端。

8. 单击**OK**按钮。**项目浏览器**窗口中将出现一个FPGA终端。

当前可在FPGA终端下添加一个FPGA VI，并开发该FPGA VI。

相关概念：

- 添加FPGA时基时钟至LabVIEW项目
- 添加FPGA终端至LabVIEW项目
- 添加项至项目浏览器窗口中的FPGA终端
- 创建FPGA I/O项
- 创建FPGA衍生时钟
- 配置FPGA时基时钟
- 控制FPGA VI的执行速率
- 下载FPGA VI至FPGA终端的闪存
- 选择用于FPGA终端的顶层时钟
- 选择用作SCTL定时源的FPGA时钟
- 使用CLIP时钟
- FPGA应用和项目简介

配置FPGA终端

通过**项目浏览器**窗口可配置FPGA终端。FPGA终端功能随具体终端变化。关于终端可用配置选项的详细信息，见指定FPGA终端的硬件文档。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**属性**。此时将显示

FPGA终端属性对话框。

4. 从**类别**列表中选择**常规**。
5. 输入FPGA终端的**名称**。**名称**用于在**项目浏览器**窗口标识FPGA终端。
6. 如添加FPGA终端至项目时未选择已有的终端，则输入用于FPGA终端的**资源**。**资源**将**项目浏览器**窗口中的FPGA终端与连到用于开发的计算、网络计算机或RT终端的指定FPGA终端关联在一起。

> **注：** 在NI Measurement & Automation Explorer (MAX)中可查看**资源**的名称。对于在MAX中显示为NI-DAQmx设备的FPGA终端，**资源**为在MAX中指定的设备名称。

7. （可选）在**执行模式**页面指定用于调试FPGA VI的执行模式。

> **注：** LabVIEW仅显示当前支持的FPGA终端的选项。

8. （可选）在**顶层时钟**页面指定顶层时钟。
9. （可选）在**组件级IP**页面添加、创建或修改组件级IP声明XML文件。
10. （可选）在**DRAM属性**页面配置用于DRAM的终端范围属性。
11. （可选）在**条件禁用符号**页面配置条件禁用符号。
12. 单击**确定**。

添加项至项目浏览器窗口中的FPGA终端

新增或现有的FPGA VI、FPGA I/O项、FPGA FIFO或FPGA时钟可被添加至项目浏览器窗口的FPGA终端。或者使用文件夹分组管理**项目浏览器**窗口中FPGA终端下的项。在要使用多个FPGA I/O项时，可能需要使用文件夹选项管理项。

按照下列步骤添加项至**项目浏览器**窗口的FPGA终端。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 右键单击FPGA终端，从快捷菜单中选择**New»x**、其中**x**为要添加的项的类型。例如，VI、FPGA I/O项或文件夹。项位于**项目浏览器**窗口的FPGA终端下。

4. 双击**项目浏览器**窗口下的新项编辑或配置项。如添加了一个FPGA时基时钟，右键单击FPGA时基时钟，从快捷菜单中选择**属性**配置时钟。

提示 提示现有项也可被拖曳至**项目浏览器**窗口中的FPGA终端。

相关概念：

- 添加FPGA终端至LabVIEW项目
- 添加监控FPGA VI的显示控件
- 添加I/O至监控FPGA VI
- 使用FPGA时钟和定时
- 选择用作SCTL定时源的FPGA时钟
- 设置和获取FPGA I/O和FPGA终端的属性
- 使用FPGA I/O节点
- FPGA应用和项目简介

查看比特文件路径

比特文件包含程序框图的全部配置信息，其中定义了FPGA的内部逻辑和数字电路，及其他与FPGA终端关联的文件中基于设备的信息。FPGA应用执行时，比特文件加载至FPGA芯片并重新配置门阵列逻辑。如要分享或引用在LabVIEW FPGA模块中创建的比特文件，需要使用比特文件路径。该路径仅可FPGA程序生成规范获取。

按照下列步骤查看对应于已编译FPGA VI的比特文件的路径：

1. 编译FPGA VI。
2. 右键单击**项目浏览器**窗口中的FPGA程序生成规范，选择**属性**。
3. 在**编译属性**对话框的**信息**页面查看VI的比特文件路径。
4. 单击**确定**。

相关概念：

- 编译、下载和运行FPGA VI
- LabVIEW FPGA编译系统

创建FPGA VI

创建FPGA VI时，请查看本节和LabVIEW文档中的主题。

相关概念：

- FPGA应用和项目简介

不支持的LabVIEW功能

受限和不可用的LabVIEW功能

- ActiveX
- 条件循环隧道
- 对话框
- 文件I/O
- 双精度或扩展精度浮点型运算
- For循环并行迭代
- 非FPGA模块专用的数学和信号处理VI
- 打印
- 编程菜单
- 共享变量（支持NI扫描引擎的部分终端除外）
- 编译时转换为单个大小的数组
- 多维数组
- VI服务器
- VI服务器的属性和方法

对其他LabVIEW功能的支持随终端变化。

多维数组或编译时不能转换为单个大小的数组

在FPGA VI中只能使用一维数组。LabVIEW必须在编译时静态确定数组的大小。如有

需要，可右键单击数组索引并从快捷菜单中选择**设置大小**，将数组常量、输入控件或显示控件设置为固定大小。该选项仅在FPGA VI中可用。

编译时LabVIEW必须转换数组为单个大小。即某些数组属性（例如，用户读取/写入元素的长度或索引）必须为常量值。例如，使用数组子集函数时，**索引**和**长度**输入必须为常量，LabVIEW才能确定输出**子数组**的设置大小。下列两种方式可选择其一：直接连线常量值至函数，或依赖常量折叠传输值。

提示 由于用户可在FPGA数组中存储大量的数据，因此可能会超出可用的FPGA资源。如需释放FPGA资源，可考虑减少FPGA应用中的数组大小。

共享变量和NI扫描引擎

对共享变量的支持随FPGA终端和RT控制器变化。并非所有的CompactRIO RT控制器都支持NI扫描引擎。关于FPGA终端和RT控制器硬件功能的详细信息，见具体FPGA终端或RT控制器的硬件文档。

支持的数据类型

FPGA VI支持下列数据类型：

- 8位有符号和无符号整数
- 16位有符号和无符号整数
- 32位有符号和无符号整数
- 64位有符号和无符号整数
- 布尔
- 定点
- 用于特定函数的单精度浮点型
- 支持的数据类型的簇
- 编译时可转换为单个大小的支持数据类型数组

相关概念：

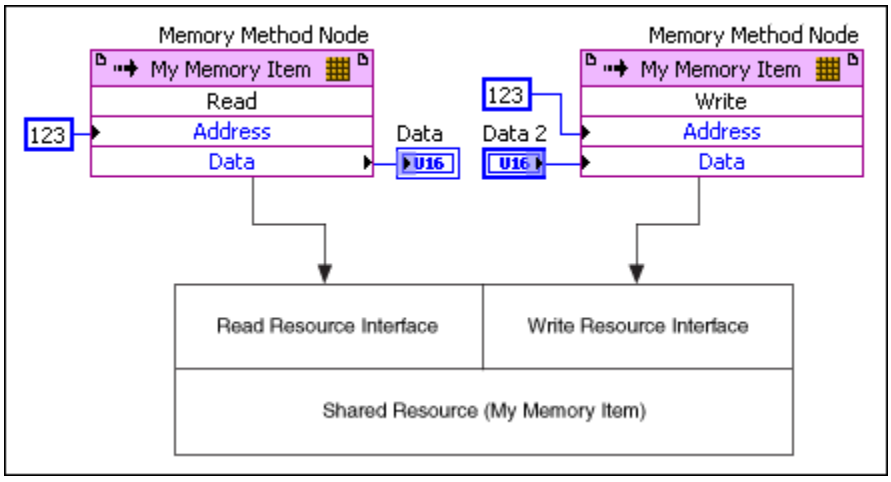
- 使用带有寄存器项、存储器项、FIFO和握手项的自定义数据类型

管理共享的资源

某些LabVIEW FPGA模块应用包含可被FPGA VI中的多个对象（例如，函数或子VI）访问的共享资源。下列为可能的共享资源：

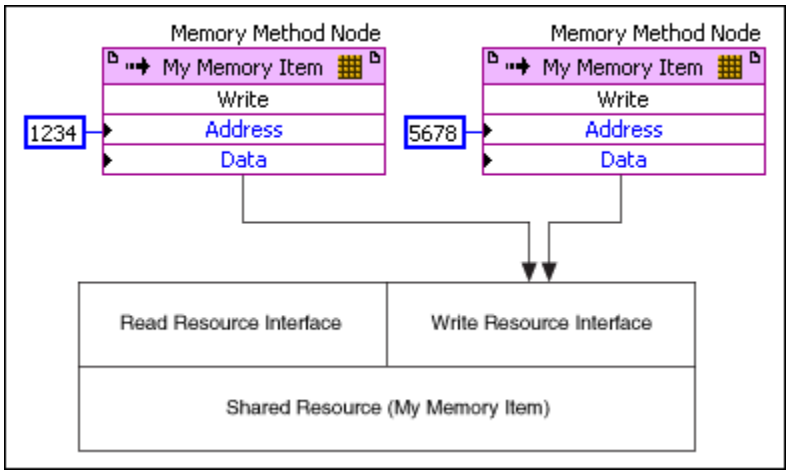
- 数字输出线
- 模拟数据线
- 寄存器项
- 存储器项
- FIFO
- 握手项
- 中断数据线
- 局部变量和全局变量
- 非重入子VI

每个共享资源包含一个或多个资源接口。资源接口在对象和共享资源间通信，如下图所示。



在FPGA VI程序框图中2个或多个对象同时请求通过同一个资源接口访问同一个共享资源将产生资源冲突。在上面的示意图中未发生资源冲突，因为对象请求通过两个不同的资源接口访问该共享资源。

但在下列示意图中，两个存储器方法节点请求通过同一个资源接口访问同一个共享资源（我的存储器项）。如两个存储器方法节点同时请求访问共享资源将产生资源冲突。

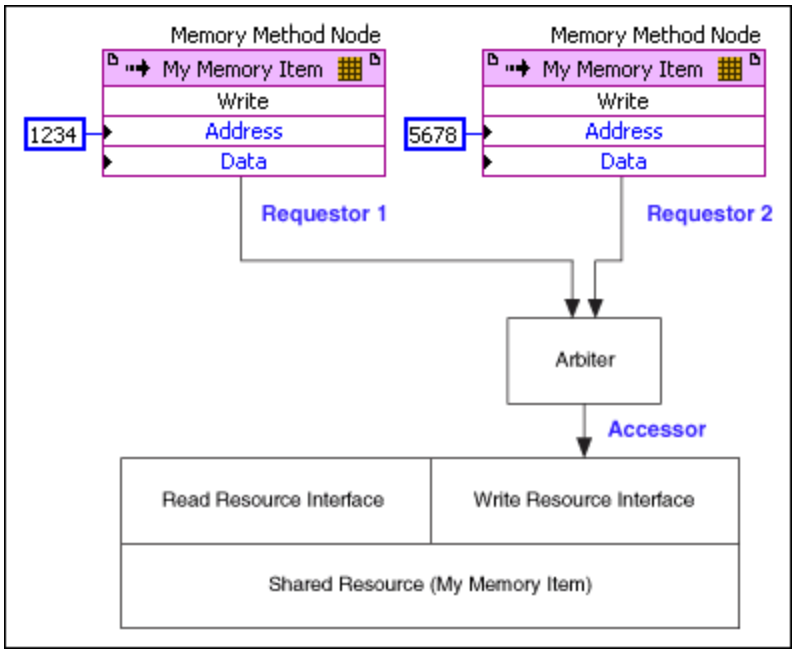


通过仲裁避免资源冲突

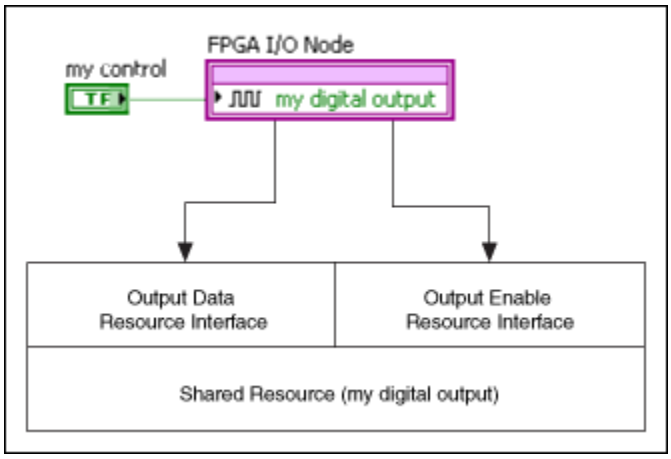
为避免资源冲突，FPGA模块提供了当多个对象同时请求访问资源时的仲裁选项，用于判定可访问资源的对象。用户选择的仲裁选项决定了LabVIEW是否使用仲裁器。可选的仲裁选项随资源变化。

注： 握手项和中断数据线不支持仲裁。

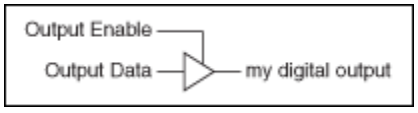
请求方是指通过资源接口请求访问资源的对象。仲裁器允许请求方对资源的访问后，该请求方转为访问方。如下图所示。



多数对象与单个资源接口交互。但在某些情况下，某些对象与多个接口交互。例如，如将FPGA I/O节点用于双向数据线，FPGA I/O节点与相应资源上的输出数据和输出启用接口交互。如下图所示。



上图基本为下列简化框图中电路的映射。



在上图中，电路分别控制**输出启用**和**输出数据**，因此LabVIEW FPGA模块将**输出启用**和**输出数据**表示为独立的资源接口。

如需要优化FPGA VI，可配置FPGA I/O、FIFO和存储器项的每个资源接口的仲裁选项。如FPGA VI设计与FPGA匹配、满足性能期望以及编译无错，可保留默认的仲裁选项。I/O的默认仲裁选项取决于FPGA终端和I/O资源。FIFO和存储器项的默认仲裁选项为**仅在有多个请求方时仲裁**。其他包括仲裁（例如，输入控件和全局变量）的资源包括固定的仲裁设置。此类资源的仲裁选项不能更改，因此在某些情况下，可能需要修改程序框图代码以避免产生资源冲突。

注： 仲裁过程可能需要几个时钟周期的执行时间。仲裁占用了额外的时间和FPGA空间，并为应用程序添加了抖动。用户可设计FPGA VI以避免抖动。

相关概念：

- 避免仲裁以优化FPGA VI
- 避免因资源冲突引起的抖动
- 使用FPGA I/O
- 判定何时使用重入或非重入子VI
- 理解仲裁选项
- 优化FPGA VI的执行速度和大小
- 同步FPGA和主机
- 使用DRAM
- 使用并行操作

理解仲裁选项

FPGA模块的可用仲裁选项如下：

- 始终仲裁
- 仅在多个请求时仲裁
- 从不仲裁

判定器在仲裁过程中执行下列通用步骤：

1. 等待一个或多个请求方。多个请求方请求访问时，通过判定器确定执行的请求方。

   注： 判定器最终确定的多个请求方的执行顺序为非确定性的。

2. 从访问方传递数据至资源接口。
3. 开始资源执行。
4. 等待资源完成执行。
5. 将数据传递回访问方。
6. 准备下一次执行的资源。
7. 等待下一个请求方。

仲裁需要使用大量的FPGA资源。如将资源接口的请求方数量在整个FPGA VI层次结构中减少为一个，可使用**仅在有多个请求方时仲裁**或**从不仲裁**仲裁选项。单个请求方不需要使用仲裁。

始终仲裁

即使仅有一个请求方请求访问，带有**始终仲裁**选项的资源接口也会一直使用仲裁器。**始终仲裁**仲裁器采用公正的轮询机制，能够确保按顺序访问共享资源。在下一个等待的请求方成为访问方之前，判定器不允许当前请求方再次成为访问方。因此，同步请求方超出一个时将产生抖动。

需要单个请求方通道与多个请求方通道同步时，选择**始终仲裁**选项。关于同步通道的信息，见启用仲裁的定时FPGA VI。

仅在多个请求时仲裁

带有**仅在有多个请求方时仲裁**选项的资源接口仅在FPGA VI包含不止一个请求方时使用仲裁器。如果资源接口具有多个请求方，即使请求方未同步，LabVIEW也会生成仲裁电路。如果共享资源在整个FPGA VI层次结构中仅包含一个请求方，使用**仅在有多个请求方时仲裁**仲裁选项可节省FPGA VI的时间和空间。

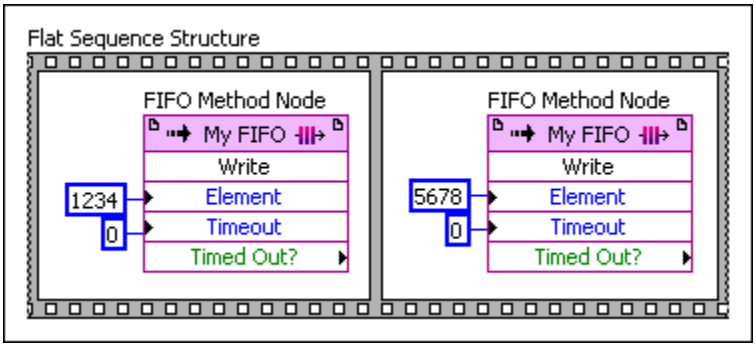
在下列情况下使用**仅在有多个请求方时仲裁**：

- FPGA VI较大且需要节省空间。
- 在整个FPGA VI层次结构中资源接口仅有一个访问方。
- 单个请求方通道不需要与多个请求方通道同步。关于同步通道的信息，见启用仲裁的定时FPGA VI。
- 在单周期定时循环内使用配置为读取或写入方法的FIFO方法节点。
- 在单周期定时循环内使用配置为读取或写入方法的存储器方法节点。
- 在单周期定时循环内使用数字输出。

从不仲裁

如果您为资源接口选择**从不仲裁**选项，则LabVIEW不会添加仲裁组件，显著节约了FPGA资源。除了节省资源，**从不仲裁**选项还使一些FPGA I/O和FIFO函数可在单时钟周期内执行。但对于多个请求方的情况，多个信号将被合并为一个信号，从而导致产生额外的逻辑占用。要使用**从不仲裁**选项，您必须保证按顺序访问FPGA VI数据流中的资源接口，如下图所示。

注： 由于所有顺序帧都在一个时钟周期内执行，无法在单周期定时循环中使用平铺式顺序结构或层叠式顺序结构来确保执行顺序。



在上图中，平铺式顺序结构确保两个FIFO方法节点同时执行，资源竞态不会发生。在这种情况下，**从不仲裁**是一个合适的选项。但如果选择了**从不仲裁**选项并发出同步请求，FPGA VI的行为将无法定义且可能发生数据冲突。

注： 为确保数据完整性，即使访问请求非同步也应避免多个对象读取或写入同一个FIFO。

仿真包含多个处理器的存储器项的FPGA应用时，选择**从不仲裁**选项可能导致错误行为。例如，应用程序包含多个写入方，每个写入方可在仿真时更新指定的存储器地址。此外，如应用程序包含多个读取方，每个读取方可在仿真时读取指定的存储器地址。

相关概念：

- 避免仲裁以优化FPGA VI
- 控制I/O上电状态
- 管理共享的资源
- 避免因资源冲突引起的抖动
- 启用仲裁的定时FPGA VI

更改仲裁选项

默认状态下，LabVIEW会对除FPGA I/O以外的所有共享资源执行仲裁。FPGA I/O默认行为取决于FPGA终端和FPGA I/O资源。如要优化FPGA VI，可自定义“项目浏览器”窗口中FPGA I/O和FPGA FIFO项的仲裁选项。默认仲裁选项随共享资源终端和I/O的类型变化。

修改FPGA I/O项的仲裁选项

按照下列步骤修改**项目浏览器**窗口中FPGA I/O项的仲裁选项。

1. 在**项目浏览器**窗口，右键单击FPGA I/O项，从快捷菜单中选择**属性**。此时将显示“FPGA I/O属性”对话框。
2. 从**类别**列表中选择**生成高级代码**。
3. 从下拉菜单中选择所需的仲裁选项。

修改FPGA FIFO项的仲裁选项

按照下列步骤修改**项目浏览器**窗口中FPGA FIFO项的仲裁选项。

1. 在**项目浏览器**窗口，右键单击FIFO项，从快捷菜单中选择**属性**。此时将显示

“FIFO属性”对话框。
2. 从**类别**列表中选择接口。
3. 从下拉菜单中选择所需的仲裁选项。

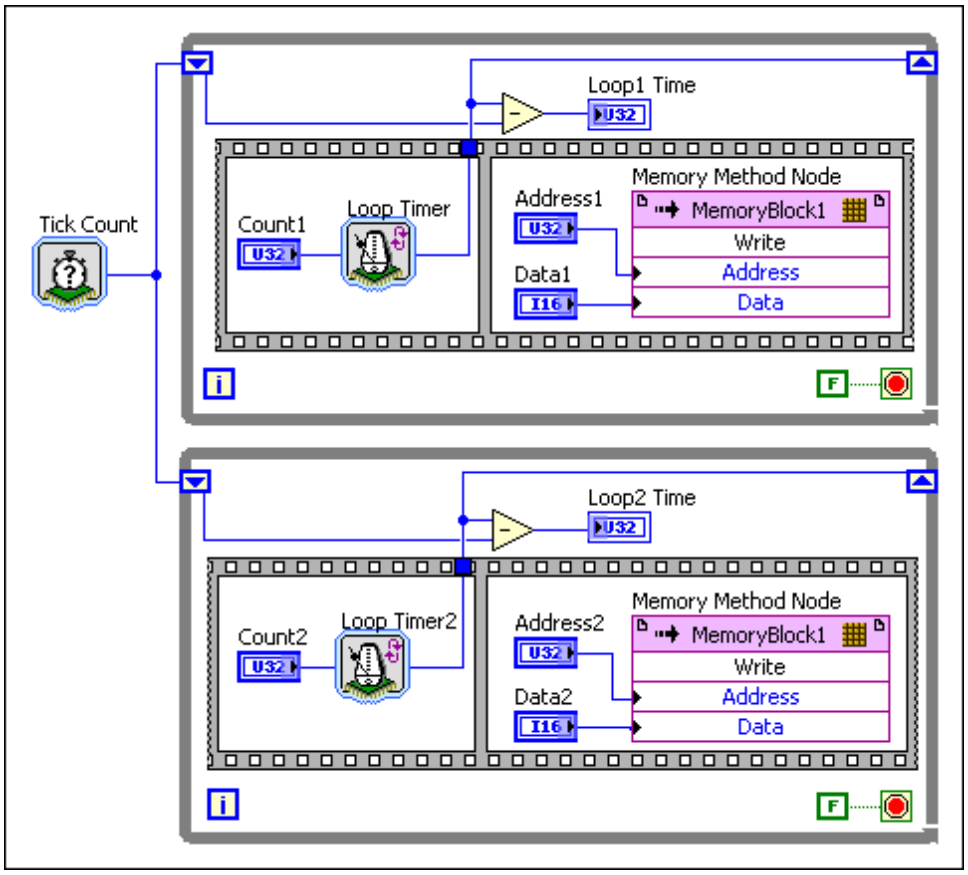
相关概念：

- 创建FPGA I/O项

避免因资源冲突引起的抖动

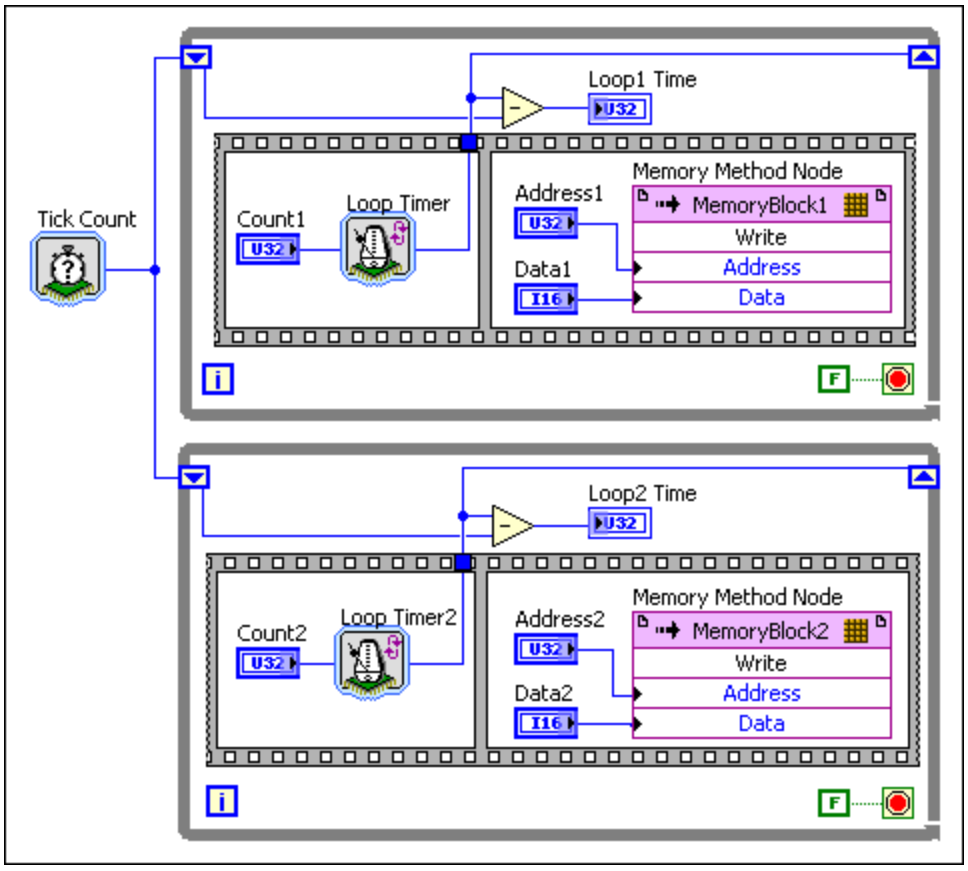
如循环中的请求方在转换为访问方时由于一个或多个其他请求方引起的资源冲突发生延迟，将会产生抖动且延时随每个循环改变。例如，假设有一个执行以固定频率采样模拟输入信号的定时While循环的应用程序。每次FPGA I/O节点执行时，节点发出模拟输入资源请求后立即成为访问方。如添加一个采样同一个模拟输入资源的While循环，这两个FPGA I/O节点可能同时请求访问模拟输入资源。此时判定器会延时其中一个访问方，并允许另一个请求方成为访问方。由于资源访问不能立即响应发出的请求，所以此延时将引入抖动。

为避免抖动，设计FPGA VI程序框图时应确保请求方不会在共享资源不可用时访问共享资源。并确保两个请求不会在同一个时钟周期内发生。在VI的两个独立部分使用共享子VI或通过并行循环访问资源接口时经常产生抖动，如下列程序框图所示。



上面程序框图中的VI包含并行的While循环，两个循环均写入**存储器块1**。根据每个循环中剩余代码需要的执行时间，两个存储器方法节点可能同时发出访问**存储器块1**的请求，导致两个循环均产生抖动并产生数据不确定性。在上述范例中，**计数1**和**计数2**间隔10～15个时钟滴答时产生抖动。

对单个资源的访问数越多，产生抖动的可能性越大。但如果能避免同时发出请求，无论潜在的访问方数量多少，判定器的延时均为常量。为避免上述范例中的多个访问方，可创建另一个存储器项并写入至底层While循环，如下列程序框图所示。

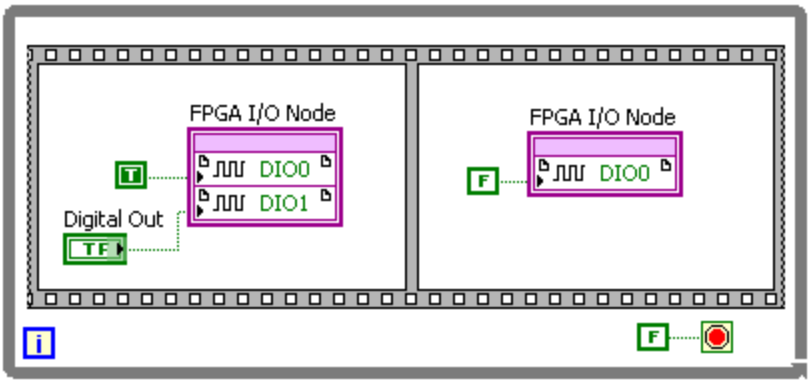


相关概念：

- 管理共享的资源
- 理解仲裁选项

启用仲裁的定时FPGA VI

并非所有的仲裁选项的执行时间都相同。如要同时访问多个同一类型的资源，必须为每个资源选择采用相同执行时间的仲裁选项。下列程序框图显示了一个可能存在定时问题的FPGA VI，实际情况取决于所选的仲裁选项。



上述程序框图中的FPGA I/O节点带有三个仲裁选项。选择**始终仲裁**选项时，LabVIEW对DIO0和DIO1执行同等的判定。两个判定器的执行时间相同，因此只要程序框图中的其他对象不同时请求访问DIO0或DIO1，DIO0和DIO1在平铺式顺序结构的第一个帧将同步输出。

选择**仅在有多个请求方时仲裁**选项时，LabVIEW将对DIO0执行判定，而不对DIO1执行判定。由于DIO0接受来自程序框图两个不同位置的访问，因此DIO0使用判定。由于DIO1仅被访问一次，因此DIO1不使用判定。DIO0的执行时间长于DIO1，因此DIO0和DIO1不支持在平铺式顺序结构的第一个帧同步输出。

如两个FPGA I/O节点均选择**从不仲裁**选项，LabVIEW不会执行DIO0或DIO1判定。因此DIO0和DIO1产生同步输出。此操作比较安全，因为顺序结构在没有其他程序框图对象请求同步访问DIO0的情况下，确保了对DIO0的顺序访问。

相关概念：

- 理解仲裁选项

使用FPGA I/O

通过FPGA终端上的输入和输出(I/O)连接FPGA终端至其他设备。FPGA I/O资源为FPGA终端的固定单元，可用于在系统的不同部分间传递数据。在某些FPGA终端上，FPGA I/O资源与前面板连线板、PXI背板或RTSI连接器对应。在其他FPGA终端上，FPGA I/O资源为FPGA内部的节点，用于连接NI设计的FPGA与用户设计的FPGA。对于组件级IP (CLIP)，I/O资源为FPGA内部的节点，用于连接CLIP和FPGA

VI。每个FPGA I/O资源均具有特定的类型，例如数字或模拟。一个FPGA终端可能具有多个相同或不同类型的I/O资源。用户可创建FPGA I/O项、判定FPGA终端上的待用I/O资源及为使用的I/O资源分配唯一的名称。

注： 关于所用FPGA终端上的I/O功能及支持特性见指定FPGA终端的硬件文档。关于仲裁I/O资源的详细信息见管理共享资源。

资源将物理量转化为用户在FPGA模块软件中操作的数字值，或将软件值转化为物理量。每个I/O资源具有一个或多个用于接收或生成物理量的接线端。许多FPGA终端上的I/O资源均具有物理接线端，用户可通过这些接线端直接连接系统单元。

FPGA VI在FPGA终端运行时在硬件中执行I/O运算。例如，如配置FPGA I/O节点读取数字线，FPGA I/O节点将读取该数字线并返回结果至FPGA VI。由于FPGA VI在FPGA上运行，VI可采用FPGA终端硬件可用的速度和确定性响应输入端。

模拟和数字输入及输出可放在程序框图的同一个节点内。FPGA I/O项的终端范围属性和方法可分别配合FPGA I/O属性节点和FPGA I/O方法节点使用。

数字I/O

FPGA终端可将数字I/O资源组织为独立的数据线或数据线组（即端口）。某些FPGA终端可访问数字I/O资源（仅为独立数据线或端口）。其他FPGA终端允许用户访问同一物理数据线（包括独立数据线和端口）。数字I/O资源分为三类：分别为读取输入、写入输出和执行上述两种函数的资源。关于数字I/O资源的支持的详细信息，见FPGA终端硬件文档。

数字输入资源

通过FPGA I/O节点可读取数字输入FPGA资源。通过数字输入资源监视FPGA终端外部的电路。

数字输出资源

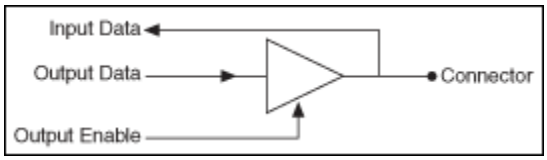
通过FPGA I/O节点可写入数字输出FPGA资源。通过数字输出资源控制FPGA终端外部的电路。

数字输入和输出资源

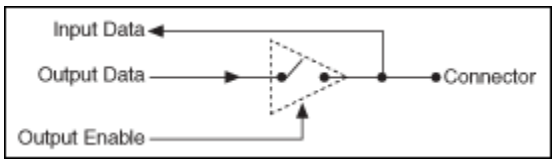
某些FPGA资源包含双向或三态的数字I/O资源。在LabVIEW FPGA中，三态数字I/O资源称为数字输入和输出资源。通过FPGA I/O节点可读取/写入数字输入和输出资源。数字输入和输出资源可用于监视和控制FPGA终端外部的电路。三态资源可用于配置I/O资源和控制数据流的方向。通过已配置“设置输出启用”方法的FPGA I/O方法节点改变数据流的方向。连线TRUE值至**启用**输出端，以配置三态I/O资源为输出资源。连线FALSE值至**启用**输出端，以配置三态I/O资源为输入资源。

注： 通常默认的方向为输入。

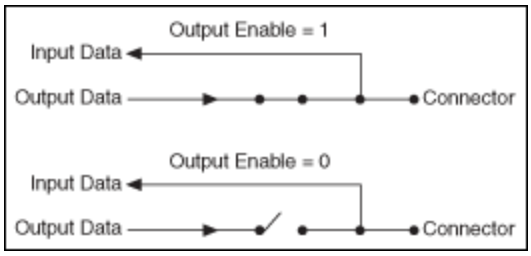
如下列示意图所示，输出启用信号控制数据线配置为输入还是输出。



三态资源用作一个由输出启用信号控制的开关，如下列示意图所示。

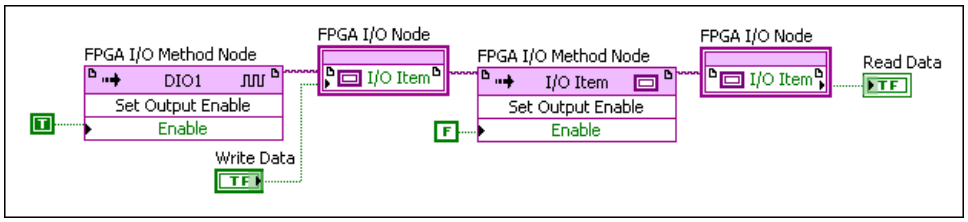


当输出启用信号为TRUE或等于1时，数据线配置为输出，且输出数据中存储的值将被驱动至数字线。当输出启用信号为FALSE或等于0时，数据线配置为高阻态输入，此时数据线可由外部设备驱动，如下列示意图所示。



无论输出启用信号的状态如何，输入数据均可用于监视数据线的当前状态。

如使用FPGA I/O节点读取数字输入和输出资源，FPGA I/O节点不会改变资源的方向。使用数字输入和输出资源写入输出时，用户使用同一资源读取输入时必须禁用该输出。如下列示意图所示，配置了一个数字输入和输出资源的FPGA I/O节点读取外部信号的状态前，带有设置输出启用方法的FPGA I/O方法节点禁用了输出数据线。否则，FPGA I/O节点仅读取FPGA VI写入的最新值。



使用FPGA I/O节点写入数字输入和输出资源时，FPGA I/O节点同时写入数据并启用输出接线端。或者使用FPGA I/O方法节点和设置输出数据方法写入数据，而无需启用输出接线端。使用FPGA I/O方法节点和设置输出启用方法启用数字接线端，此操作允许数据被驱动输出。在设置输出启用方法前使用设置输出数据方法，以在启用输出时指定数字输出和输出资源的状态。例如，程序框图的某一部分可能持续生成内部信号。当内部信号实际被驱动至一个外部设备时，在程序框图的另一部分使用FPGA I/O方法节点和设置输出启用方法进行独立控制。

如在单周期定时循环内包含了数字I/O资源，对应于单周期定时循环的每次计数，每个同步寄存器均将引入延迟。在某些情况下，FPGA外部的延迟可能对系统影响较大。如LabVIEW框图和FPGA间的延时的准确模型对于通过在开发计算机上执行FPGA VI测试应用的逻辑非常重要，可延时I/O仿真数据为对I/O节点的调用数量，其等于同步寄存器的数量。

模拟I/O

模拟输入

如所用的FPGA终端包含模拟输入资源，用户必须配置FPGA I/O节点以读取模拟输入的值。如配置FPGA I/O节点读取一个模拟输入，FPGA I/O节点可能初始化转换、等待结果并返回电压的二进制表示值（有符号整数或定点数）。模拟输入过程和输出数据类型随FPGA终端变化。对于多数FPGA终端，创建FPGA VI，以将模拟输入FPGA I/O节点返回的数据用于FPGA内部的运算。数据也可被传回主控VI，且可能的情况下，如FPGA终端的模拟输入端连接了传感器，可将数据转换为电压或其他物理量。

用于转换二进制表示法为物理量的公式取决于FPGA终端和传感器。例如，对于NI PXI-7831R设备，使用下列公式转换二进制表示值为电压：输入电压 = （二进制编码/ 32768) × 10.0V。更多信息见FPGA终端硬件的文档。

模拟输出

如所用的FPGA终端包含模拟输出资源，用户可配置FPGA I/O节点以写入一个模拟输出值。如配置FPGA I/O节点写入数据至模拟输出，FPGA I/O节点将电压的二进制表示值写入至数模转换器(DAC)，该转换器将设置模拟输出电压。数据类型随FPGA终端变化。电压有两种生成方式－主控VI或FPGA VI。通常主控VI写入值至FPGA VI前，先转换电压为恰当的二进制表示值。如FPGA VI判定电压，通常FPGA VI使用相应的二进制表示法执行计算。在上述两种情况下，DAC生成一个对应二进制表示值的电压。

用于转换电压至二进制表示值的公式取决于指定FPGA终端。例如，对于NI PXI-7831R设备，使用下列公式转换电压为二进制表示值：二进制编码 = （输出电压 × 32768) / 10.0V。更多信息见FPGA终端硬件的文档。

相关概念：

- 管理共享的资源
- 使用并行操作

创建FPGA I/O项

如要访问FPGA终端上的I/O资源，首先必须在项目浏览器窗口中创建一个FPGA I/O项。然后可在FPGA VI的程序框图上使用FPGA I/O节点的FPGA I/O项。必须在**项目浏览器**窗口的FPGA终端下创建FPGA I/O项。

提示 多数FPGA终端支持FPGA项目向导，通过该向导可快速创建用于终端的I/O。

如添加一个C系列模块至**项目浏览器**窗口，LabVIEW将自动添加所有FPGA I/O项至项目。如添加组件级IP (CLIP)项至**项目浏览器**窗口，LabVIEW自动添加所有与CLIP关联的I/O项至项目。

按照下列步骤新建一个FPGA I/O项。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 右键单击FPGA终端，从快捷菜单中选择**新建»FPGA I/O**。此时将显示新建FPGA I/O对话框。
4. 在**可用资源**目录树中选择要用的I/O资源。

   提示 按下<Ctrl>键的同时单击其他资源，可同时选择多个资源。在可用的情况下单击顶层I/O类型或文件夹，可选择整个I/O资源组。

5. 单击**添加**按钮。I/O资源将出现在**新建FPGA I/O**表的**资源**一栏。I/O资源的默认名称将出现在**名称**一栏。在**名称**栏可输入新的I/O资源名称。
6. 重复步骤4和5，直至所有想用的I/O资源出现在**新建FPGA I/O**表中。
7. 单击**OK**按钮。创建的FPGA I/O项将出现在**项目浏览器**窗口的FPGA终端下。如未取消勾选**新建FPGA I/O**对话框的**在文件夹中放置新建I/O**复选框，FPGA I/O项将位于文件夹中。文件夹组织结构取决于**可用资源**列表的架构。

在程序框图上通过FPGA I/O节点也可创建FPGA I/O项。右键单击FPGA I/O节点，从快捷菜单中选择**添加新的FPGA I/O**，显示**新建FPGA I/O**对话框。按照上述步骤4至

步骤7，在**项目浏览器**窗口的FPGA终端下添加新的FPGA I/O项。

> **注：** 添加CLIP项至项目时，与组件级IP (CLIP)关联的I/O将出现在**项目浏览器**窗口中。

相关概念：

- 添加FPGA终端至LabVIEW项目
- 添加I/O至监控FPGA VI
- 更改仲裁选项
- 使用FPGA I/O节点
- 在FPGA I/O和FPGA终端上执行操作
- 设置和获取FPGA I/O和FPGA终端的属性

配置FPGA I/O项

可在“项目浏览器”窗口或程序框图的FPGA I/O节点内配置FPGA I/O项。用户在**项目浏览器**窗口中指定的配置选项将被应用于FPGA终端上的FPGA I/O项。在FPGA I/O节点中配置的选项仅应用于FPGA节点中的FPGA I/O项，而不是整个FPGA终端。

在项目浏览器窗口配置FPGA I/O项

按照下列步骤配置**项目浏览器**窗口的FPGA I/O项。

1. 右键单击**项目浏览器**窗口中的FPGA I/O项，从快捷菜单中选择**属性**，显示“FPGA I/O属性”对话框。
2. 从**类别**列表中选择**常规**，为FPGA I/O项分配名称。
3. 配置**类别**列表下显示的其它选项。

> **注：** FPGA I/O配置选项因I/O资源和FPGA终端而异。某些FPGA I/O项不能被配置。其它FPGA I/O项具有几个配置选项。从**类别**列表中选中一个选项并单击**帮助**按钮，了解FPGA I/O项可用的配置选项。

配置FPGA I/O节点中的FPGA I/O项

按照下列步骤配置FPGA I/O节点中的FPGA I/O项。

1. 右键单击FPGA I/O节点，从快捷菜单中选择**属性**，显示“FPGA I/O节点属性”对话框。
2. 配置**类别**列表中FPGA I/O项下显示的选项。

> **注：** FPGA I/O配置选项因I/O资源和FPGA终端而异。某些FPGA I/O项不能被配置。其它FPGA I/O项具有几个配置选项。从**类别**列表中选中一个选项并单击**帮助**按钮，了解FPGA I/O项可用的配置选项。

使用FPGA I/O节点

在项目浏览器窗口中创建FPGA I/O项后，可在程序框图的FPGA I/O节点中使用FPGA I/O项。可在**项目浏览器**窗口单击FPGA I/O项并拖曳其至程序框图，以创建一个新的配置了该FPGA I/O项的FPGA I/O节点。或者按照下列步骤添加FPGA I/O节点至程序框图并配置用于指定FPGA I/O项的接线端。

1. 新建一个VI或打开一个FPGA终端下已包含FPGA I/O项的已有VI。
2. 添加一个FPGA I/O节点至程序框图。
3. 单击FPGA I/O节点的元素部分添加一个新的FPGA I/O项或选中一个之前添加至项目的FPGA I/O项。可选择所需的任意FPGA I/O项，无需考虑I/O资源的类型。新的FPGA I/O项位于FPGA I/O节点中。通过连线FPGA I/O输入控件至FPGA I/O节点的**FPGA I/O In**输入端可编写可重用代码。

展开FPGA I/O节点可添加额外的FPGA I/O项。右键单击FPGA I/O节点中的FPGA I/O项，从快捷菜单中选择**添加元素**。新的**I/O项**接线端位于FPGA I/O节点中。然后重复步骤3配置新的**I/O项**接线端。

> **提示** 使用定位工具单击节点的上沿或下沿并上下拖曳边沿，也可展开FPGA I/O节点。LabVIEW会自动按照FPGA I/O项在**项目浏览器**窗口中的顺序

> 填充每个额外的接线端。展开节点前可先在**项目浏览器**窗口中调整FPGA I/O项的顺序。在**项目浏览器**窗口，选择FPGA终端下的FPGA I/O项然后拖曳其至想要放置的位置。

相关概念：

- 创建FPGA I/O项
- 添加项至项目浏览器窗口中的FPGA终端

在FPGA I/O和FPGA终端上执行操作

FPGA I/O方法节点可用于在FPGA I/O项和C系列模块中调用方法或动作。在某些情况下，也可在FPGA终端上调用方法。可用的方法由选定的FPGA终端、FPGA I/O项或C系列模块确定。

按照下列步骤创建和配置FPGA I/O方法节点。

1. 添加FPGA I/O方法节点至程序框图。
2. 右键单击FPGA I/O方法节点，从快捷菜单中选择**选择项**。**选择项**菜单将显示FPGA终端、FPGA I/O项和**项目浏览器**窗口的FPGA终端下的C系列模块。选择所需的项。

   > **提示** 或者，也可以连线FPGA I/O输入控件至FPGA I/O方法节点的**FPGA I/O输入**输入端，以编写可重用代码。

3. 右键单击FPGA I/O方法节点，从快捷菜单中选择**选择方法**，选择需要分配FPGA I/O项的方法。LabVIEW在**选择方法**菜单中显示FPGA终端的可用方法。如FPGA终端不支持选中项的方法，LabVIEW将在快捷菜单中显示**无可用的方法**。关于FPGA终端支持方法的信息，见具体的终端硬件文档。

   > **提示** 也可单击FPGA I/O方法节点，从快捷菜单中选择FPGA终端的可用方法。

4. 从快捷菜单中选择方法后，LabVIEW在FPGA I/O方法节点中显示方法和方法接线

端。连线所需的接线端。

> **提示** FPGA终端方法节点是为项目中的FPGA终端配置的FPGA I/O方法节点。

相关概念：

- 创建FPGA I/O项

设置和获取FPGA I/O和FPGA终端的属性

FPGA I/O属性节点可用于编程获取LabVIEW项目中的FPGA I/O项和C系列模块的属性。在部分情况下，在FPGA终端本身也可以设置并获取属性。可用的属性取决于FPGA终端和项目浏览器窗口中与FPGA I/O项关联的I/O资源。

按照下列步骤创建和配置FPGA I/O属性节点。

1. 新建一个VI或打开FPGA终端下的现有VI。
2. (可选)添加FPGA I/O项至项目。
3. 添加FPGA I/O属性节点至程序框图。
4. 右键单击FPGA I/O属性节点，从快捷菜单中选择**选择项**。**选择项**菜单将显示FPGA终端、FPGA I/O项和**项目浏览器**窗口的FPGA终端下的C系列模块。选择所需的项。

   > **提示** 或者连线FPGA I/O输入控件至FPGA I/O属性节点的**FPGA I/O输入**的输入端，以编写可重用代码。

5. 右键单击FPGA I/O属性节点的**属性**接线端，从快捷菜单中选择**选择属性**，以选择要分配给项的属性。LabVIEW在**选择属性**菜单中显示FPGA终端的可用属性。如FPGA终端不支持选中的项的属性，LabVIEW将在快捷菜单中显示**无可用的属性**。关于FPGA终端支持的属性的详细信息，见具体FPGA终端的硬件文档。

   > **提示** 用户也可单击FPGA I/O属性节点，从快捷菜单中选择FPGA终端

的可用属性。

6. （可选）右键单击FPGA I/O属性节点，从快捷菜单中选择**添加元素**添加额外的**属性**接线端。使用定位工具单击节点的上沿或下沿并上下拖曳边沿，也可展开FPGA属性节点。

提示 FPGA终端方法节点是为项目中的FPGA终端配置的FPGA I/O方法节点。

相关概念：

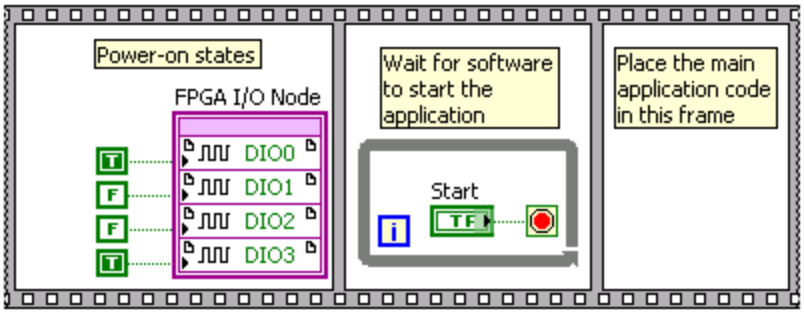
- 创建FPGA I/O项
- 添加项至项目浏览器窗口中的FPGA终端

控制I/O上电状态

系统上电时，应用可能需要FPGA终端的I/O设置为已知的值。例如，如果FPGA终端通过数字输出控制液压阀，FPGA终端必须保持关闭这些阀门，直至主控VI启动并开始控制系统。可创建一个FPGA VI并配置FPGA终端，以设置FPGA终端的上电状态。

注： 如要实现本主题涉及的技术，必须使用带有闪存的FPGA终端。

必须对FPGA VI进行编程，以便程序框图无需依赖主控VI便可设置输出状态。例如，可在顺序结构的第一个帧内放置数字和模拟输出函数。然后将余下的LabVIEW代码放在顺序结构的后续帧内，如以下程序框图中所示。然后配置FPGA VI在其被载入FPGA后立即执行。编译FPGA VI并将其下载至FPGA终端的闪存，然后配置FPGA终端在FPGA终端上电时，自动从闪存载入FPGA VI。FPGA终端上电后，FPGA VI将从闪存载入到FPGA，然后立即开始执行。FPGA VI顺序结构的第一个帧内的输出函数将设置上电输出状态。



不仅可为FPGA终端的输出创建静态上电状态，还可创建执行复杂操作的任意上电功能。例如，可根据输入状态设置输出状态，与外部设备进行串行通信等。关于默认上电状态的详细信息见指定FPGA终端的硬件文档。

注： 如上电状态后仅使用I/O资源一次，可选择**从不仲裁** 仲裁选项，以节省空间。

相关概念：

- 理解仲裁选项

创建触发器和计数器

FPGA模块包含执行基本I/O的函数。但是，用户可能有需要自定义I/O功能的应用。将FPGA I/O节点作为构件来创建自定义I/O应用（例如，触发器和计数器）。

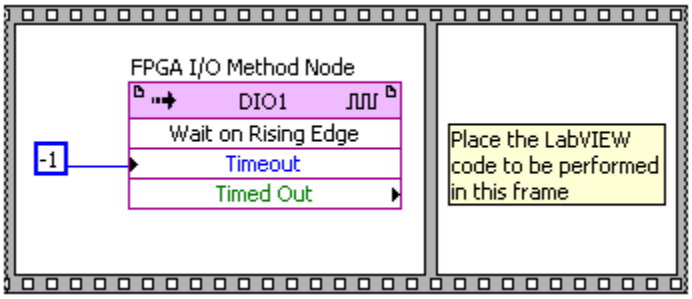
创建触发器

在多数应用中，执行操作前均需要等待触发。使用FPGA I/O方法节点的等待上升沿方法，可在单个数字输入端等待触发。

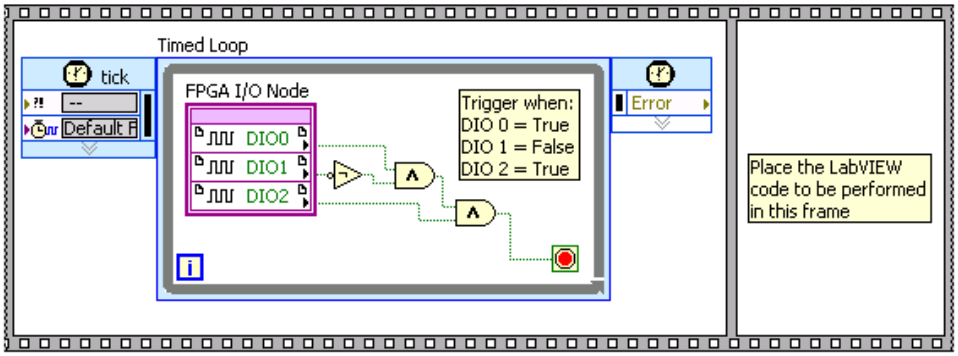
注： 可用的I/O资源及相关方法随终端和配置变化。关于可用方法和I/O资源的信息，见具体的FPGA终端硬件文档。

“等待上升沿”方法等待直至数字输入端满足任一用户指定的条件。在顺序结构的第一个帧内放置FPGA I/O方法节点，在下一个帧内放置用于该任务的LabVIEW代码，

如以下程序框图所示。



FPGA I/O节点可创建更高级的触发事件。例如，仅在多个数字线满足给定条件时执行触发的应用。如下列程序框图所示。

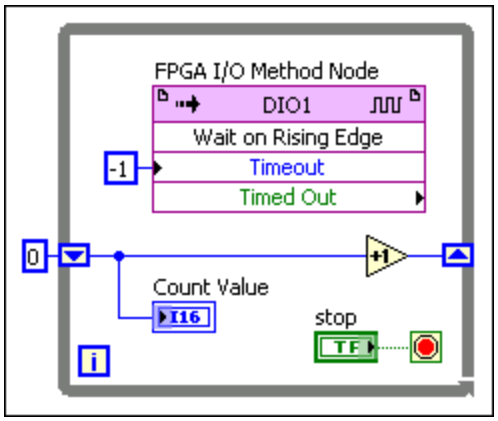


在单周期定时循环内可放置配置了数字输入资源的FPGA I/O节点，且仅当数字输入满足触发器模式时退出单周期定时循环。在顺序结构的第一个帧内放置单周期定时循环。这个操作与上个范例中对等待上升沿方法的等待操作类似。

使用While循环可以同样的方式执行模拟触发器。在While循环中放置一个配置了模拟输入I/O资源的FPGA I/O节点和比较函数，当模拟输入的值满足可编程条件时触发。

创建计数器

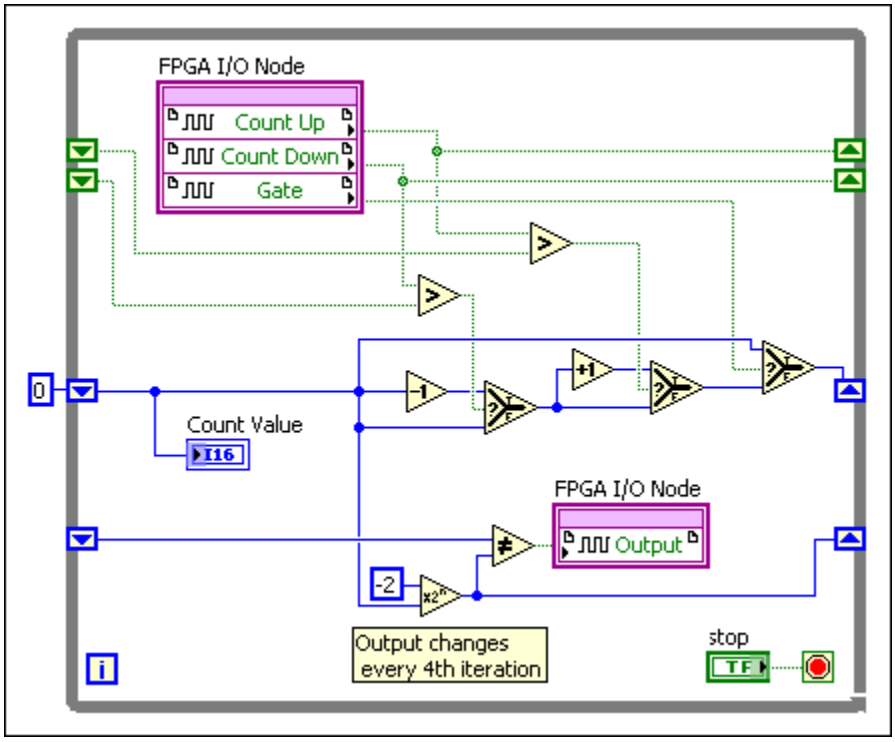
计数器可以是简单的事件计数器，也可以是带有多个输入和输出端的复杂信号测量计数器。在While循环中使用FPGA I/O方法节点函数可创建简单事件计数器。例如，可使用等待上升沿方法等待数字输入接线端出现上升沿，如以下程序框图所示。



FPGA I/O方法节点检测到边沿后，程序框图的计数器值加1，且将计数器的值存储在While循环的移位寄存器内。使用前面板显示控件或局部变量可查看计数器的值。

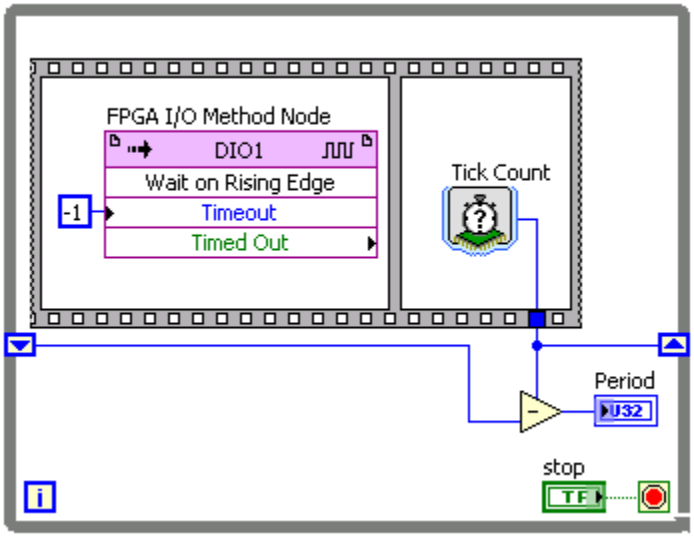
注： 可用的I/O资源及相关方法随终端和配置变化。关于可用方法和I/O资源的信息，见具体的FPGA终端硬件文档。

FPGA I/O函数可创建更高级的计数器。例如，应用可能需要带有独立向上计数、向下计数及门输入和输出的计数器，如以下程序框图所示。



在上述程序框图中，当**向上计数**产生上升沿时，计数器的值加1。当**向下计数**产生上升沿时，计数器的值减1。当**门**为高电平时，**门**将阻止向上计数和向下计数改变计数器的值。**向上计数**、**向下计数**和**门**为FPGA终端的指定数字I/O资源名称。计数器的值为4的倍数时输出值无效。在LabVIEW代码中可使用简单布尔判定确定计数器为升值计数、减值计数或保持不变。或者使用算术判定确定输出值是否无效。

FPGA I/O方法节点可用于测量输入信号，如以下程序框图所示。例如，用户可能需要测量输入信号的周期。可以在顺序结构的第一个帧内放置FPGA I/O方法节点，第二个帧内放置时间计数器Express VI。然后放置该顺序结构至While循环内。连线时间计数器VI返回的当前值至移位寄存器，以用作下一次执行的While循环的输入值。然后从当前时间中减去之前的时间，确定输入信号的周期。



提示使用单周期定时循环可增加执行速率，减少计数器应用中的FPGA使用量和抖动。

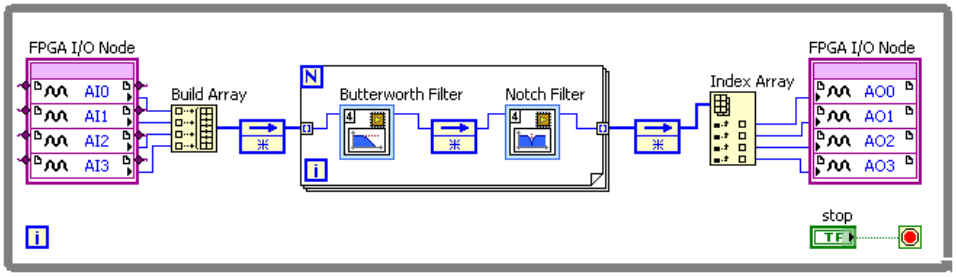
使用多个输入通道滤波FPGA I/O

使用FPGA数学与分析选板上的滤波VI滤波FPGA I/O。默认状态下，FPGA滤波VI可使用单输入通道。但可使用滤波VI配置对话框的**通道数**选项，配置用于多输入通道的VI。下列滤波VI支持多个输入通道：

- Butterworth滤波器
- 陷波滤波器

- 有理分式重采样

配置用于多个输入通道的FPGA滤波VI时，必须发送每个输入通道至队列中的滤波VI。用户可捆绑输入通道至一个数组，并使用For循环确保滤波VI接收队列中的每个输入通道，如下列程序框图所示。



在上一个程序框图中，4个模拟输入信号通过Butterworth滤波器和一个陷波滤波器。为最大化吞吐量，VI可使用反馈节点创建一个流水线。

注： 部分FPGA终端支持IO采样方法，该方法可提供来自队列中多个输入通道的数据。如终端支持IO采样方法，无需捆绑输入通道至数组。但For循环内需包含配置了IO采样方法的FPGA I/O方法节点。

部分支持多通道的滤波VI也支持多通道握手机制。

相关概念：

- 使用流水线优化FPGA VI
- 使用握手信号控制定时

使用FPGA时钟和定时

FPGA应用中可使用下列时钟：

- **时基时钟**—终端硬件中的数字信号，可用作FPGA应用的时钟。
- **衍生时钟**—通过可用作FPGA应用时钟的时基时钟创建的时钟。使用衍生时钟缩放FPGA终端时基、外部或CLIP时钟的频率。

- **顶层时钟**—在单周期定时循环外部的用于FPGA VI的全局时钟。

LabVIEW设置FPGA VI编译过程中生成的电路的定时限制时，使用时基时钟属性。

放置在FPGA VI中的每个VI或函数均需要一定的执行时间，称为逻辑延时。FPGA终端上的顶层时钟决定了FPGA VI程序框图上的每个函数和VI的执行时间。如改变了顶层时钟的频率，程序框图上函数的执行速率和FPGA VI的执行速率也会发生改变。

通过控制FPGA VI的执行速率，可指定FPGA应用的定时需求。如未包含额外的编程，运算在由VI的数据流确定的速率发生。如要控制或测量执行定时可使用定时VI。定时VI还可用于创建自定义I/O应用（例如，计数器和触发器）。

相关概念：

- 添加FPGA时基时钟至LabVIEW项目
- 添加项至项目浏览器窗口中的FPGA终端
- 配置FPGA时基时钟
- 创建FPGA衍生时钟
- 选择用作SCTL定时源的FPGA时钟
- 使用CLIP时钟

FPGA VI的定时考虑因素

如要使用数据流模型执行代码，LabVIEW将同步FPGA上的逻辑。默认情况下，LabVIEW FPGA在程序框图的逻辑函数间放置一个寄存器，以最大化每个操作执行所需的传播时间。

传播延时是指信号由一个寄存器传播至下一个寄存器所需的时间。组合路径是信号由一个寄存器经另一个寄存器的逻辑和接线的集合。

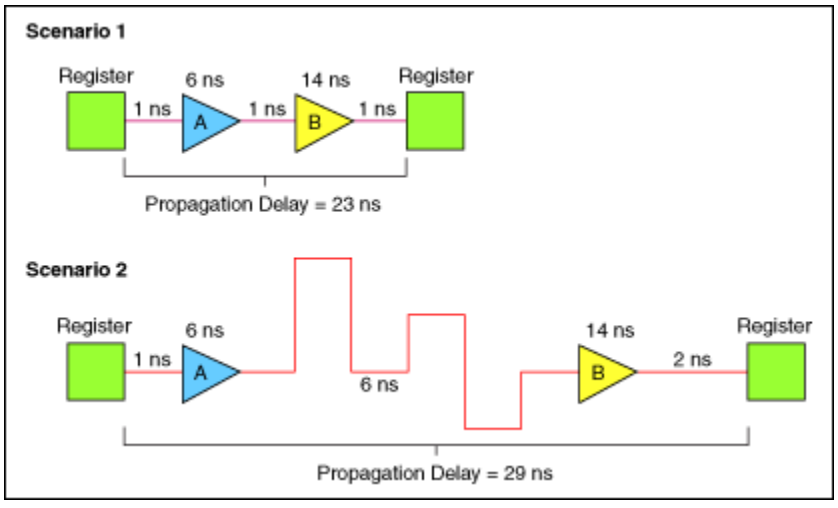
由于每个时钟周期均更新寄存器内容，因此传播延时必须小于时钟周期。传播延时由两部分组成：逻辑延时和连线延时。逻辑延时是信号经过的逻辑门数量和类型的函数，通常表示传播延时中最重要的组件。连线延时是信号通过的连线路径的函数，其通常很小。因为FPGA编译器尝试尽可能紧密的集合组合路径的组件。但当

FPGA VI达到FPGA的容量限制时，函数间的物理间隔将增加。同时连线延时将成为两个寄存器间总体传播延时的主要组成部分。如两个寄存器间的传播延时大于FPGA时钟速率，FPGA编译器将返回定时错误。该定时错误称为时间限制或周期限制冲突。

注： 给定函数的逻辑延时随终端变化。连线延时在每次FPGA VI编译时均发生变化。

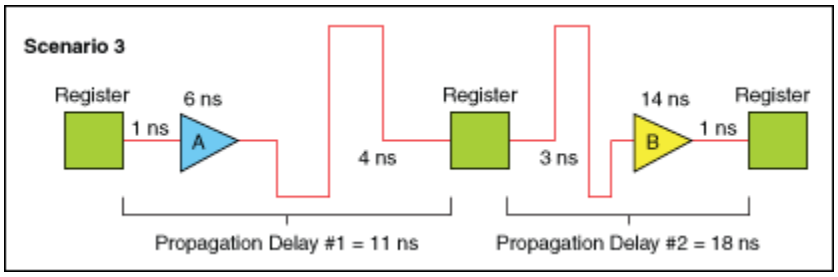
LabVIEW FPGA模块专用于生成可在单周期定时循环外部以至少40 MHz的时钟速率运行的电路。40 MHz时钟速率对应的是25 ns的时钟周期。为了避免2个寄存器间的传播延时超出25 ns，多数LabVIEW函数均包含一个输出寄存器，因此需要一个完整的时钟执行周期。如2个寄存器间的传播延时超出了25 ns，FPGA VI无法在40 MHz默认的时钟速率完成编译。

例如，假设函数A需要6 ns的逻辑延时，函数B需要14 ns的逻辑延时。如顺序连线函数A和B，而不在函数间添加寄存器。整体逻辑延时为20 ns。如要在40 MHz的默认时钟速率下完成编译，连线延时只能为5 ns。根据FPGA编译器在函数间的连线方式，连线延时可能超出或不超出5 ns。如下列场景和1和2所示。



在场景1中，设计满足40 MHz的定时限制。在场景2中，设计不满足40 MHz的时钟限制。当用户尝试编译FPGA VI时将产生定时冲突错误。相比之下场景3中，2个函数之间添加了寄存器。添加寄存器后将产生2个独立的传播延时。即使连线路径较

长，2个传播延时均可实现在40 MHz内完成编译。



当函数位于单周期定时循环外时，FPGA编译器将在函数的逻辑级间均匀放置寄存器，以将逻辑划分为可在默认FPGA时钟速率内执行的部分。如函数包含运行在FPGA上的内部寄存器（例如，存储器方法节点），函数的执行时钟周期数量与函数中寄存器的数量相等。

如在同一时钟速率下需要以较小的延时执行逻辑，可使用单周期定时循环。如在单周期定时循环内放置函数，编译器不会包含用于该函数的输出寄存器，因此单周期定时循环可在一个时钟周期内完成。某些函数（例如按窗函数缩放或FFT Express VI）即使在位于单周期定时循环内时，也需要多个时钟周期。握手机制可用于管理上述函数的数据定时。

如单周期定时循环内的传播延时大于时钟周期，定时冲突分析窗口将指出未能满足定时要求的单周期定时循环。在某些情况下，可使用反馈节点或移位寄存器缩短组合路径的长度，以实现流水线设计。

注： 如在单周期定时循环内使用高吞吐率数学函数，可添加内部寄存器，缩短函数间的组合路径长度。

大型FPGA应用的资源考虑

每个FPGA终端包含一定数量的触发器。由于寄存器使用触发器，用于FPGA VI的寄存器数量和类型将决定FPGA VI是否能够满足FPGA终端的需求。通常寄存器使用的触发器数量与数据类型宽度相对应。例如，布尔寄存器仅需要一个触发器来存储数据，而一个I64寄存器需要使用64个寄存器来存储数据。

对于多数用户来说，FPGA上有限的触发器数量并不是问题。但如FPGA上的空间耗

尽，则必须优化FPGA VI的大小。

相关概念：

- FPGA硬件概述
- 使用流水线优化FPGA VI
- 缩短FPGA VI的组合路径
- 使用单周期定时循环优化FPGA VI
- 使用握手信号控制定时

选择用于FPGA终端的顶层时钟

在项目浏览器窗口可定义用于FPGA终端的顶层时钟。顶层时钟是位于 单周期定时循环外部的用于FPGA VI的全局时钟。FPGA终端默认使用任意FPGA时基时钟。按照下列步骤选择用于FPGA终端的顶层时钟。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 如所用的FPGA终端未自动添加FPGA时基时钟至**项目浏览器**窗口，则添加FPGA时基时钟。如要使用FPGA时基时钟作为顶层时钟的定时源，请忽略下列步骤。
4. （可选）创建一个FPGA衍生时钟。
5. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**属性**。此时将显示FPGA终端属性对话框。
6. 在**类别**列表中选择**顶层时钟**。
7. 如要使用默认的FPGA终端时钟，选择**默认**。如要使用配置时钟，在**配置时钟**列表中选择**选择配置时钟**和一个已配置的时钟。
8. 单击**OK**按钮。

注： 某些FPGA终端不允许特定的配置时钟用作顶层时钟。

相关概念：

- 添加FPGA时基时钟至LabVIEW项目

- 控制FPGA VI的执行速率
- 创建FPGA衍生时钟
- 添加FPGA终端至LabVIEW项目
- 选择用作SCTL定时源的FPGA时钟

更改顶层FPGA终端的时钟速率

每个FPGA终端提供至少一个用于控制FPGA内部操作的时钟。FPGA终端时钟能够确定FPGA VI程序框图上单个VI和函数的执行速率。使用较高的时钟速率编译FPGA VI可实现更高的性能。但并非所有FPGA VI都能在较高的时钟频率下进行正确编译。如选择的时钟速率相对于FPGA VI来说过快，编译状态窗口将显示由于定时冲突，编译失败。必须修复定时冲突并重新尝试编译。

在项目浏览器窗口右键单击FPGA终端，从快捷菜单中选择**属性**，更改用于FPGA终端的顶层FPGA终端时钟速率。在FPGA终端属性对话框的顶层时钟页面设置顶层时钟。在FPGA VI内双击**输入节点**，在配置定时循环对话框中选择时钟速率，更改单周期定时循环的时钟速率。时钟可选择FPGA终端顶层时钟或来自FPGA终端时基的时钟。

在FPGA VI中更改顶层FPGA终端时钟速率或单周期定时循环的时钟速率后，必须重新编译FPGA VI。

相关概念：

- 定时冲突的疑难解答
- 在单周期定时循环中执行代码
- 实现多个时钟域

添加FPGA时基时钟至LabVIEW项目

FPGA项目可支持不只一个时基时钟，即使某些终端在用户添加该终端时，仅添加了支持的时基时钟的一个子集至项目浏览器窗口。如添加终端至LabVIEW项目时，FPGA终端未包含在**项目浏览器**窗口所需的FPGA时基时钟，用户可添加一个新的时基时钟至LabVIEW项目。

按照下列步骤添加一个新的时基时钟至LabVIEW项目。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 右键单击FPGA终端，从快捷菜单中选择**新建»FPGA时基时钟**。这时会出现FPGA时基时钟属性对话框。
4. 从**资源**下拉菜单中选择可用的时钟。
5. 单击**OK**按钮。新的FPGA时基时钟将出现在**项目浏览器**窗口的FPGA终端下。

当前可在项目中设置新的时基时钟为顶层时钟。

相关概念：

- 使用FPGA时钟和定时
- 添加FPGA终端至LabVIEW项目
- 选择用于FPGA终端的顶层时钟
- 配置FPGA时基时钟
- 创建FPGA衍生时钟
- 选择用作SCTL定时源的FPGA时钟

配置FPGA时基时钟

通过指定FPGA时基时钟的名称和特定属性，可配置与FPGA终端关联的FPGA时基时钟。可用的属性随FPGA终端和选中的FPGA时基时钟变化。按照下列步骤配置与FPGA终端关联的FPGA时基时钟。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 如所用的FPGA终端未自动添加FPGA时基时钟至项目浏览器窗口，则添加FPGA时基时钟。
4. 在**项目浏览器**窗口，右键单击FPGA时基时钟，从快捷菜单中选择**属性**。这时会出现FPGA时基时钟属性对话框。
5. 选择要配置的选项。
6. 单击**OK**按钮。

相关概念：

- 使用FPGA时钟和定时
- 添加FPGA终端至LabVIEW项目
- 添加FPGA时基时钟至LabVIEW项目
- 控制FPGA VI的执行速率

创建FPGA衍生时钟

在LabVIEW项目中可通过FPGA时基时钟衍生额外的时钟。按照下列步骤衍生一个FPGA时钟。

> **注：** FPGA衍生时钟的支持随FPGA终端变化。更多信息见FPGA终端硬件的文档。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 如所用的FPGA终端未自动添加FPGA时基时钟至项目浏览器窗口，则添加FPGA时基时钟。
4. 在**项目浏览器**窗口，右键单击FPGA时基时钟，从快捷菜单中选择**新建FPGA衍生时钟**。这时会出现FPGA衍生时钟属性对话框。

   > **注：** 如FPGA终端不支持FPGA衍生时钟或添加的FPGA时基时钟，**新建FPGA衍生时钟**在快捷菜单中将显示为灰色。

5. （可选）在**名称**文本框中输入名称。默认情况下，名称为实际应用的衍生频率。
6. 在**所需衍生频率**文本框中输入所需的衍生时钟运行频率。LabVIEW将与之最接近的频率并在**实际衍生配置**部分的**衍生频率**文本框中显示。如FPGA支持**衍生频率**，则**消息**对话框将显示该频率可用。否则将显示衍生频率和所需频率之间的误差。
7. 单击**确定**按钮接受该衍生频率并关闭对话框。FPGA衍生时钟将出现在**项目浏览器**窗口中，并带有衍生频率的缩写。

当前可在项目中设置新的衍生时钟为顶层时钟。

相关概念：

- 添加FPGA终端至LabVIEW项目
- 控制FPGA VI的执行速率
- 使用FPGA时钟和定时
- 添加FPGA时基时钟至LabVIEW项目
- 选择用于FPGA终端的顶层时钟
- 选择用作SCTL定时源的FPGA时钟

控制FPGA VI的执行速率

对于某些FPGA终端，用户可配置FPGA时基时钟并在项目中设置为顶层时钟以控制执行速率。或者使用衍生时钟克服时基时钟的配置限制。不同的FPGA终端支持不同的FPGA衍生时钟。关于不同的FPGA终端的可用的时基时钟配置的详细信息，见指定FPGA终端的硬件文档。

按照下列步骤创建一个FPGA衍生时钟，以控制程序框图上的项的执行速率。

注： 如使用时基时钟或衍生时钟作为单周期定时循环的定时源，则单周期定时循环内部的代码以时基时钟或衍生时钟速率执行。如配置FPGA时基时钟或创建一个衍生时钟，并设置为顶层时钟，可在单周期定时循环外部控制代码的执行速率。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 创建FPGA衍生时钟以在预期的执行速率运行。
4. 设置FPGA衍生时钟为顶层时钟。

注： 并非全部FPGA VI具有相同的最大时钟速率。FPGA VI的复杂程度可影响FPGA终端的最大执行速率。如选择的时钟速率相对于FPGA VI来说过

高，定时冲突分析窗口将返回不能符合定时要求的函数和组件的信息。

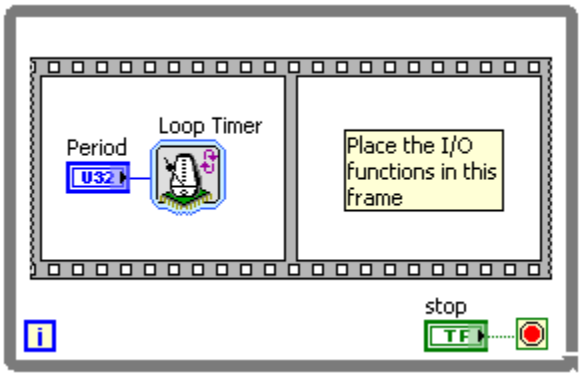
相关概念：

- 配置FPGA时基时钟
- 选择用于FPGA终端的顶层时钟
- 添加FPGA终端至LabVIEW项目
- 创建FPGA衍生时钟

使用FPGA定时函数管理执行速率

创建定时I/O应用

应用通常需要I/O在指定的频率上执行。例如，用于控制循环的算法通常需要输入信号在已知速率下被采样。在While循环中使用循环定时器Express VI控制I/O的执行速率，如下列程序框图所示。



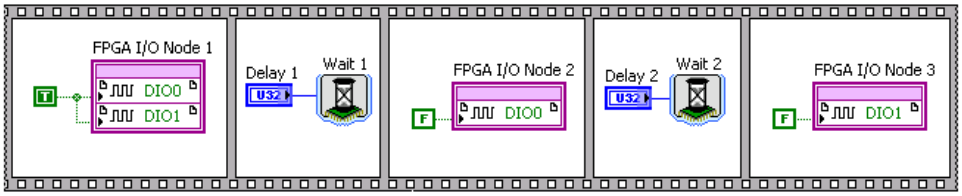
如要使用“循环定时器”Express VI控制I/O的执行速率，可在While循环内部放置一个“顺序”结构。在“顺序”结构的第一个帧内放置“循环定时器”Express VI。在弹出的**配置循环定时器**对话框中配置**计数器单位**和**内部寄存器的大小**。在”顺序“结构的后续帧内放置I/O的LabVIEW代码。

提示 选择适用于应用程序的最小的**内部寄存器的大小**，能够节省FPGA终端上的空间。

“循环定时器”Express VI的初次调用不会产生任何等待或延时，因为该调用已为后续的调用创建了引用时间标识。初次调用“循环定时器”Express VI后，必须等待**计数**参数满足指定的时间间隔要求后才会返回后续调用。如**计数**参数指定的时间小于FPGA终端所需在While循环中执行代码的时间，“循环定时器”Express VI不会影响While循环的定时。

在事件间创建延时

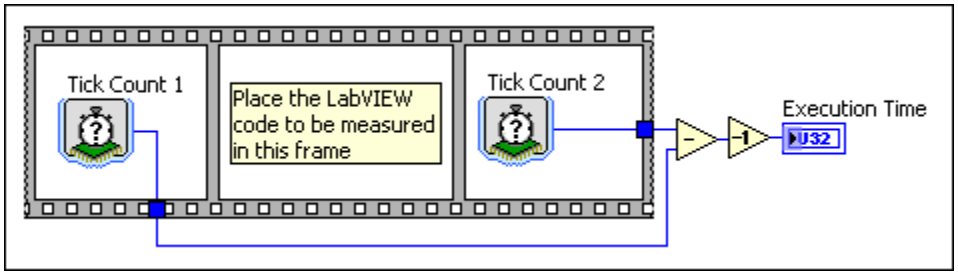
使用“等待”Express VI创建FPGA VI的事件间延时。例如，需要在触发器和后续输出间创建一个延时。用于触发器的LabVIEW代码可放置在顺序结构的第一个帧内。然后在后续的帧内放置“等待”Express VI。最后，在顺序结构的最后一个帧内放置用于输出的LabVIEW代码。或者使用多个“等待”VI在一个顺序结构内创建一组延时，如下列程序框图所示。



测量事件间隔时间

使用“时间计数”Express VI测量诸如数字信号的边沿上的事件的间隔时间。需要判定周期、脉冲宽度、输入信号的频率或要判定一组LabVIEW代码的执行时间时，可使用时间计数Express VI。

例如，如要判定函数执行时间或一组LabVIEW代码的执行时间，可使用带有两个“时间计数”Express VI的“顺序”结构，如下列程序框图所示。



在顺序结构的第一个帧内放置时间计数Express VI。然后在顺序结构的第二个帧内放置要测量的LabVIEW代码。最后，在顺序结构的最后一个帧内放置另一个“时间计数”Express VI。然后可通过计算两个“时间计数”Express VI的结果差值得出代码的执行时间。从计算结果中减去1，以补偿“时间计数”Express VI的执行时间。

时间计数Express VI带有一个内部计数器来记录时间。在同一程序框图内放置的每个时间计数Express VI的内部计数器均使用同一开始时间。因此，使用同一**计数器单位**值和同一**内部计数器的大小**选项的每个时间计数Express VI记录同一时间。例如，如同时调用两个使用同一**配置时间计数**选项的“时间计数”Express VI，它们将返回相同的**时间计数**的值。

“时间计数”Express VI返回以**计数器单位**为单位的整数值。**时间计数**的值不能表示分数时间周期，此类时间可能出现在**计数器单位**配置为**uSec**或**mSec**的情况下。配置**计数器单位**为**uSec**或**mSec**可能导致定时测量具有±1 **计数器单位**值的精度误差。例如，在上述程序框图中配置时间计数Express VI测量以毫秒为单位的时间。如第一个“时间计数”Express VI的执行时间为47.9毫秒，**时间计数**返回的值为47。如第二个“时间计数”Express VI的执行时间为53.2毫秒，**时间计数**返回的值为53。尽管此范例的时间延时为5.3毫秒，但返回值之间的差值为6毫秒。

相关概念：

- 使用整型数据类型

测量While循环的执行速率

按照下列步骤在FPGA VI中测量While循环的执行时间。

1. 在程序框图上放置一个While循环。
2. 右键单击While循环的条件接线端，根据应用程序需求从快捷菜单中选择**创建常量**或**创建输入控件**。
3. 添加要在While循环内执行的代码。
4. 添加时间计数器Express VI至While循环，在**配置时间计数器**对话框中单击**确定**按钮。
5. 右键单击While循环，从快捷菜单中选择**添加移位寄存器**。

6. 添加减函数至While循环。
7. 将时间计数器Express VI的**时间计数器**接线端连线至While循环的右侧移位寄存器和“减”函数的**x**接线端。
8. 连线左侧的移位寄存器至“减”函数的**y**接线端。
9. 右键单击“减”函数的**x–y**接线端，选择**创建**»**显示控件**。
10. 运行VI。**x–y**显示控件中显示的值即为While循环的执行速率。

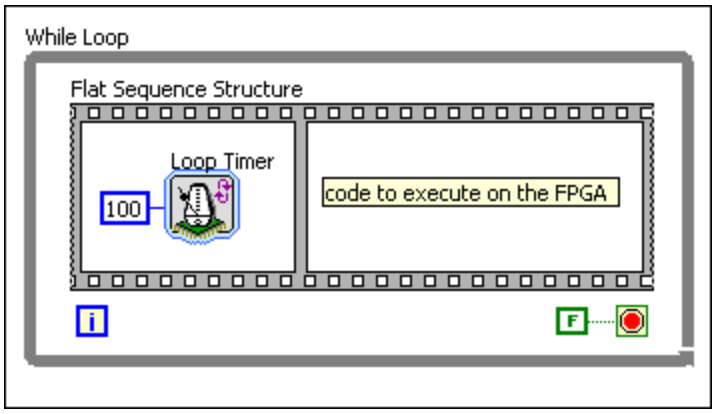
强制VI在指定时间间隔内执行

按照下列步骤强制FPGA VI中的循环在特定的时间间隔执行。

1. 在程序框图上放置一个While循环。
2. 右键单击While循环的条件接线端，根据应用程序需求从快捷菜单中选择**创建常量**或**创建输入控件**。
3. 在While循环内添加平铺式顺序结构。
4. 在平铺式顺序结构中添加循环定时器Express VI。
5. 在配置循环定时器对话框中配置循环定时器，然后单击**确定**按钮。
6. 右键单击循环定时器Express VI的**计数**输入接线端并选择**创建**»**常量**。输入While循环计数间所需的时间间隔。
7. 右键单击“平铺式顺序结构”，从快捷菜单中选择**在后面添加帧**。
8. 添加要在FPGA中执行的代码至平铺式顺序结构的新建帧。

> **注：** 代码执行时间必须少于步骤6中指定的时间。如果代码的执行时间长于指定的时间，执行时间将决定循环频率，并覆盖步骤6中指定的循环频率。

以下程序框图显示了使用平铺式顺序结构指定FPGA VI定时的方法。



在单周期定时循环中执行代码

在FPGA VI中可使用单周期定时循环优化代码、实现多个时钟域及在默认FPGA终端时钟或用户指定时钟的一个时钟周期内执行代码。

放置代码至单周期定时循环内部后，LabVIEW不会在编译函数代码中放置启用链寄存器。这将增加代码的组合路径长度，且编译FPGA VI时可能导致定时冲突错误。

注： LabVIEW不能移除在存储器方法节点和FFT Express VI函数内部的寄存器。

由于通过单周期定时循环内部逻辑的最长路径增加了，最大时钟速率降低了。用户可流水线较长的组合路径，以保持最终的最大时钟速率较高。或者可将独立的逻辑部分拆分为不同的时钟域。即可在较慢的时钟域内使用长的组合路径及在较快的时钟域内使用短的组合路径。

相关概念：

- 使用单周期定时循环优化FPGA VI
- 实现多个时钟域
- 更改顶层FPGA终端的时钟速率
- FPGA VI的数据流和启用链
- 缩短FPGA VI的组合路径
- 使用流水线优化FPGA VI

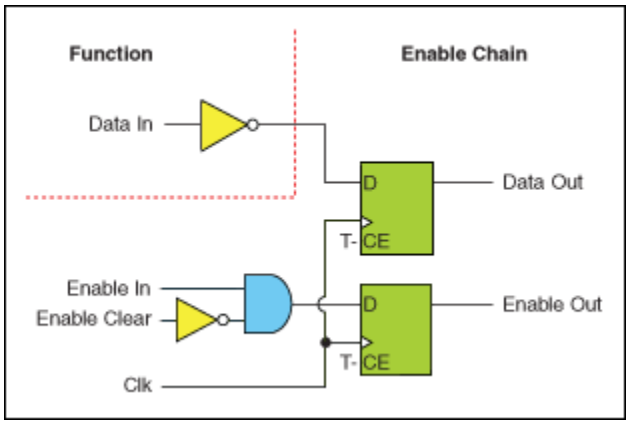
FPGA VI的数据流和启用链

启用链是LabVIEW添加至FPGA代码的附加逻辑，用于强制执行FPGA的数据流。启用链包含两个部分：

- 与程序框图上的实际数据流并行运行的一组寄存器。启用链的此部分仅位于单周期定时循环的外部。
- 强制执行单周期定时循环内部数据流的隐式启用信号该信号扇出包含状态保持元素的循环内部的全部节点。

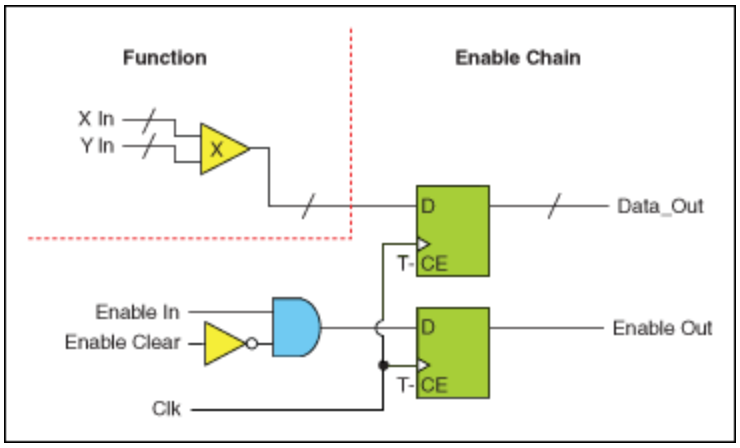
单周期定时循环外的数据流

LabVIEW以数据流的方式执行代码。当数据在节点运行所需的全部输入端就绪时，节点开始执行。节点执行结束后，节点输出传送数据至下一节点。下列示意图显示了一个要在单周期定时循环外部实现布尔函数的FPGA硬件的范例。



启用链确保FPGA逻辑按照其在LabVIEW程序框图中的顺序执行。示意图显示了函数位于单周期定时循环外部时，布尔函数转换为FPGA逻辑的过程。下列示意图的函数部分显示了程序框图上相对于“取负数”函数的布尔逻辑。启用链部分包含额外的同步寄存器，仅在时钟的上升沿输出数据。

下列示意图给出了一个执行算法操作的FPGA硬件的范例。



根据启用链的系统开销，每个函数或VI至少占用一个时钟周期。某些函数（例如，模拟输入运算）将占用几百个时钟周期。具体取决于运算的复杂程度及硬件限制。

单周期定时循环内的数据流

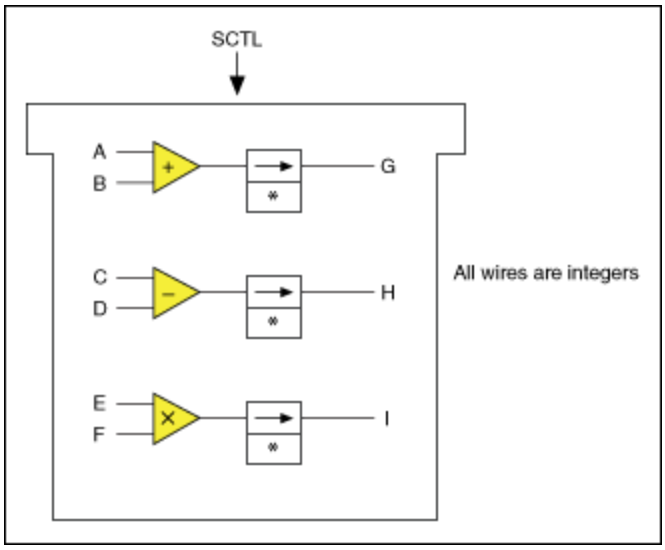
放置在单周期定时循环内部的节点不包含启用链的寄存器部分。如排除启用链的系统开销，FPGA的整体空间使用量将有所下降。因为不再需要用于启用链的触发器。同样由于不再存在启用链，单周期定时循环内的所有运算均可在一个时钟周期内完成。但某些节点（例如，存储器方法节点或FFT Express VI）需要大于一个时钟周期的执行时间，因此该节点的输出要等待至单周期定时循环的下一次迭代时才有效。获取有效数据所需的时钟周期数量为节点的延时。

隐式启用信号对定时性能的影响

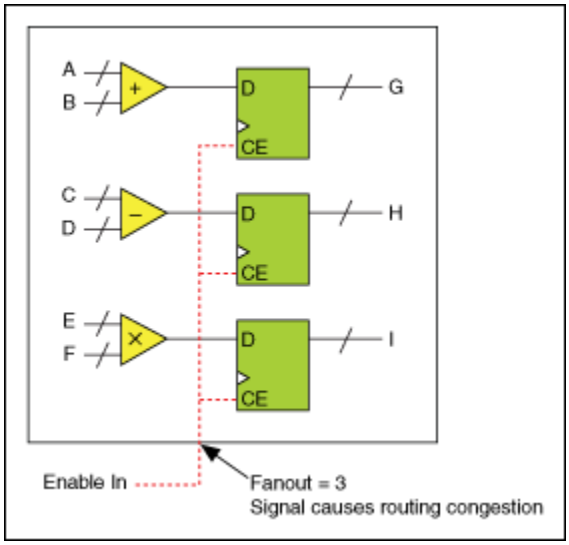
单周期定时循环内的隐式启用信号扇出程序框图内的全部触发器。LabVIEW将门控与单周期定时循环相关的时钟，直至隐式启用信号被移除。该扇出的额外连线系统开销将限制大型设计的定时性能。如设计包含独立于程序框图上其他节点运行的单周期定时循环，请考虑从上述循环中移除隐式启用信号以改善定时性能。

注： 移除隐式启用信号的支持随终端变化。关于移除隐式启用信号支持的详细信息，见终端硬件文档。

下文描述了一个包含单周期定时循环代码的程序框图。

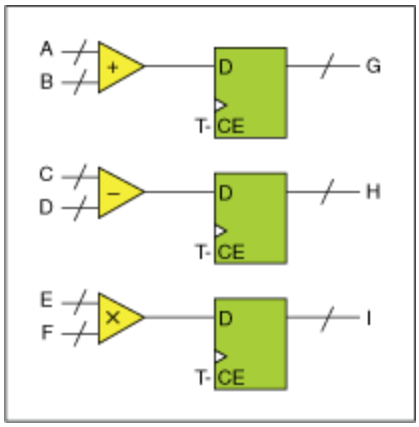


下列示意图为上述程序框图代码的VHDL表示。



注意，默认情况下，隐式启用信号扇出包含状态保持元素（例如，触发器或存储器块）的单周期定时循环的每个节点。该连线的系统开销限制了某些应用的定时性能。

下列示意图为移除了隐式启用信号的同一VHDL表达。



相关概念：

- FPGA硬件概述
- 在单周期定时循环中执行代码
- 在单周期定时循环中同步I/O
- 使用波形查看器查看信号

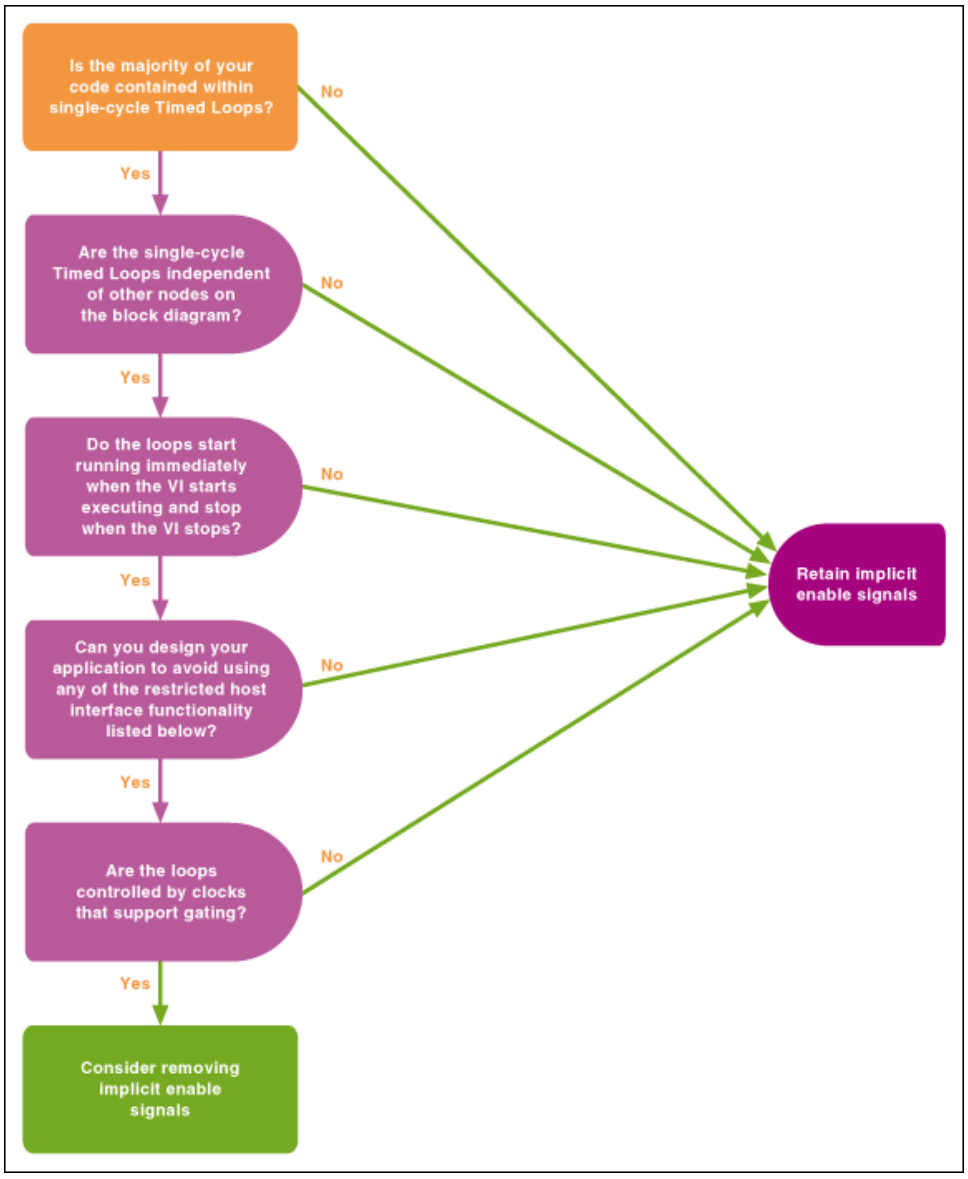
改进大型设计的定时性能

由于其添加额外的信号至FPGA逻辑，隐式启用信号将引起连线繁忙及限制定时性能。对于某些应用，应允许LabVIEW从单周期定时循环中移除独立于设计中的其他节点运行的隐式启用信号，以降低连线繁忙。

通过移除隐式启用信号受益的应用

- 高吞吐率应用
- 带有单周期定时循环的应用程序，循环运行的时钟速率较快并包含大量的代码
- 添加额外代码后应用程序编译失败，且额外代码本身满足必需的时钟速率。

使用下列流程图帮助判定应用程序是否可用于移除隐式启用信号。



受限的功能

对于移除隐式启用信号的应用，LabVIEW不支持使用下列方法和功能：

- 重置（调用方法）
- 关闭并重置
- 中止（调用方法）
- 未重新下载VI就重新运行VI
- VI开始执行前，IP经过一个隐式同步重置
- 重置无效时，IP需要运行时时钟

- VI运行前访问输入控件、显示控件或DMA
- 仅当程序框图中不包含必须在受影响的循环前或后执行的逻辑时，移除隐式启用信号。

从符合条件的循环移除隐式启用信号

按照下列步骤，允许编译器从不具有数据依赖性并独立运行的单周期定时循环内移除隐式启用信号。

1. 在编译属性对话框的信息页面，勾选**允许在单周期定时循环内移除隐式启用信号**复选框。

> **注：** 如未出现**要移除隐式启用信号**复选框，终端下打开的VI不支持移除隐式启用信号。

从全部循环移除隐式启用信号

按照下列步骤请求编译器从单周期定时循环内尝试移除隐式启用信号。如编译器不能从单周期定时循环内移除隐式启用信号，完成该步骤将导致LabVIEW返回错误。

1. 双击单周期定时循环的输入节点，显示配置定时循环对话框。
2. 勾选**需要移除隐式启用信号**复选框。

启用每个CLIP时钟的隐式信号移除

通过添加所需标识符以确保时钟支持声明XML文件的门控，必须启用该功能以在设计中移除用于每个CLIP时钟的隐式启用信号。

使用握手信号控制定时

握手是指建立了用于持续通信的参数的两个节点间的通信。对于给定程序框图的单周期定时循环中的节点F，握手机制确定发生下列动作的时间：

- F 丢弃来自上方数据流节点的数据。上方数据流节点是指发送数据至F的节点。

- F 接收来自上方数据流节点的数据。
- 下方数据流节点丢弃来自F的数据。下方数据流节点是指接收来自F的数据的节点。
- 下方数据流的节点接收来自F的数据。

在单周期定时循环中，握手是非常必要的。因为多个周期节点需要不止一个周期来计算有效的数据，但单周期定时循环强制这些节点在每个时钟周期内返回数据。因此，多周期节点不能在每个时钟周期返回有效的数据。为了确保算法的数值精确性，依赖于此数据的节点必须了解该数据为有效数据还是无效数据。

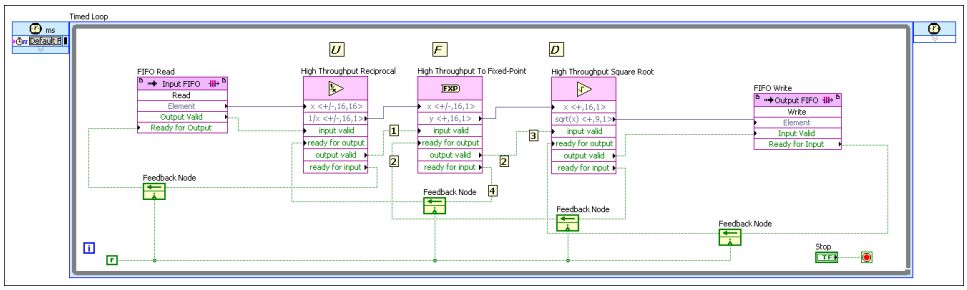
NI已经建立了可与单周期定时循环内的特定节点配合使用的握手协议。协议包含下列接线端：

- **输入有效**－指定下一个数据已经到达并等待处理。
- **输出有效**－指示节点当前处理的数据有效，且准备用于下方数据流节点。
- **准备输出**－指定数据流节点是否接收新的数据。
- **准备输入**－指示节点在下一个时钟周期是否接收新的数据。

注： 由于该协议包含4个接线端。FPGA VI中的握手机制常被称为4线协议。

确保单输入节点使用有效的数据

假设在单周期定时循环中有3个单输入函数：上方数据流函数U、函数F及下方数据流函数D。这些函数的输入值为x，并在指定时钟周期后产生输出数据。下图给出了使用NI建议的握手协议连线U、F和D的方法。



在上图中，连线端的数值取决于下列步骤：

1. U发送有效数据至F时，F的**输入有效**接线端的值变为TRUE。TRUE表示U已经产生了有效的输出值，并已发送该值至F。如要使用该逻辑，连线U的**输出有效**接线端至F的**输入有效**接线端。即F能够获知U何时产生有效的输出值。
2. F的**输入有效**接线端接收到TRUE值后，F开始计算结果。F计算结果时，F的**输出有效**接线端返回FALSE。当下列2个标准均满足时，**输出有效**接线端返回TRUE。
   - 自F开始计算结果以来，至少已经过L_f个时钟周期，其中L_f是F的延迟时间。在函数的配置对话框中可查看函数的延迟。双击函数可打开函数配置对话框。
   - F的**准备输出**接线端为TRUE。该接线端通知F，何时D已经准备好从F接收输入值。如要使用该逻辑，可通过反馈节点将D的**准备输入**接线端连线至F的**准备输出**接线端。由于反馈节点默认情况下需要一个执行时间周期，F将在D发送数据后的一个时钟周期接收到该值。
3. 如果将F的**输出有效**接线端连线至D的**输入有效**接线端，D就会知道F何时生成了有效值，并将该值发送给D。此逻辑与步骤1中的逻辑类似。除F变为上面数据流函数外，D执行的进程均相同。
4. 如在下一个时钟周期，F可接收另一个有效输入数据点，F的**输入就绪**接线端会返回TRUE。如要使用该逻辑，通过反馈节点连线F的**准备输入**接线端至U的**准备输出**接线端。即U可获知F何时准备好接收新的输入值。由于反馈节点默认情况下需要一个执行时间周期，U将在F发送数据后的一个时钟周期接收到该值。

提示如要获取高吞吐率，请在F的**吞吐率**输入控件中输入较小的值。F接收到有效输入并开始计算结果后，在F可接收下一个有效输入前至少已经经过了T_f个时钟周期，其中T_f为F的**吞吐率**控件值。因此，**吞吐率**控件的较小的值意味着相对于较大的**吞吐率**值，F能够更快的接收到下一个有效输入数据。小**吞吐率**值还意味着相对于较大的**吞吐率**值，F的**准备输入**接线端能够更早的返回TRUE。

确保多输入节点使用有效的数据

“高吞吐率加”函数为多输入函数，因此此函数有2个输入端**x**和**y**。在单周期定时循环中，必须确保在同一个时钟周期内的**x**和**y**均有效。如仅一个输入端有效，另一个输入端无效。函数将使用一个有效输入值和一个无效输入值计算输出数据。

为了确保有效x和y在同一个时钟周期内到达，给所有为x提供数据的节点添加延迟。然后给为y提供数据的所有节点提供延迟。2个延迟的总和必须相等。如果延迟总和不同，可为延迟短的路径（具有最低延迟总和的路径）添加反馈节点，直至x和y路径的延迟总和相等。

> **注：** 默认情况下，反馈节点为路径增加一个时钟周期的延迟。但用户可修改每个反馈节点的延迟。同一路径中的节点具有不同数量的流水线级时，上述设计方法很有效。

关于演示如何在多输入握手应用中平衡延迟路径的范例，见`labview\examples\target_type\FPGA Fundamentals\FPGA Math and Analysis\High-Throughput Math\Vector Normalization\Vector Normalization.lvproj`，其中target_type为`CompactRIO`或`R Series`，具体取决于所安装的驱动程序。

支持握手机制的节点

下列节点支持握手接线端：

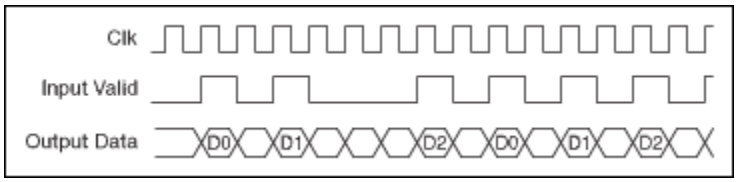
- FFT Express VI
- 握手方法节点
- 高吞吐率数学函数，除基本元素函数外。
- 线性代数函数
- 配置DRAM时的存储器方法节点
- FIFO方法节点，已启用握手接口
- 有理重采样 Express VI
- 缩放窗口 Express VI
- 用户控制I/O采样函数

支持多通道握手的节点

下列节点支持多通道握手：

- 有理重采样 Express VI

对于多通道节点，LabVIEW交错分配值，进而第一个有效数据进入第一个通道，第二个有效数据进入第二个通道，以此类推。如在通道扫描过程中，**输入有效**变为FALSE，对应下一个通道的下一个有效值立即进入队列。如下列示意图所示。



在示意图中，d0、d1和d2为3个接收数据的通道。注意每次**输入有效**接线端变为FALSE，然后又转换为TRUE时，对应下一个通道的下一个有效数据立即进入队列。

确定最快的可用吞吐率

对于单周期定时循环内部的一系列连接节点，最快吞吐率是指可发送更多数据至系列节点的最少时钟周期数。最快吞吐率等于系列中最慢的节点的吞吐率（即带有最大值**吞吐率**输入控件的节点）。此节点为所有节点的瓶颈。如系统以高于此最快吞吐率的速率向节点发送数据（即发送数据至节点时，系统等待的时钟周期少于最小允许的时钟周期数量），LabVIEW将丢弃该数据。

如要判定节点的最快吞吐率，首先查看该节点的**吞吐率**输入控件确定每个节点的吞吐率。双击节点或显示节点的即时帮助窗口可查看控件信息。具有最大值（即最慢的节点）的**吞吐率**输入控件为系统可发送数据至此系列函数的最快速率（即最少的时钟周期数）。

例如，假设一系列节点中，最慢节点吞吐率为32周期/样本，FPG终端A的时钟速率为40MHz，输入采样率为2MS/s。时钟速率除以输入采样率得到系统吞吐率，即40000000周期/ 2000000样本（20周期/样本）。此时，系统每20个时钟周期发送数据至节点，该周期比最快吞吐率32个时钟周期要快。这将导致LabVIEW丢弃数据。

注： 系统吞吐率与系统中的通道数量成反比，且通道吞吐率与系统中的通道量成正比。下列公式说明了系统吞吐率和通道吞吐率之间的关系。系统吞吐率＝通道吞吐率/通道数 通道吞吐率＝系统吞吐率X通道数。例如，系统吞吐率为200周期/采样，系统带有4个通道，则每个通道的吞吐率为

> 800周期/采样。

相关概念：

- 使用多个输入通道滤波FPGA I/O
- 优化FPGA VI的执行速度和大小
- FPGA VI的定时考虑因素
- 使用DRAM
- 在FPGA VI中交互AXI IP

实现多个时钟域

单周期定时循环可在不同时钟域下运行。在FPGA VI中使用单周期定时循环指定时钟域。所有可用的FPGA终端时钟在**项目浏览器**窗口中显示为FPGA终端时基或衍生时钟。FPGA终端支持的FPGA终端时基或衍生时钟可被用作单周期定时循环的定时源。但在单周期定时循环中放置的代码的组合路径长度必须足够短，以实现在指定的时钟周期内运行。

同步不同时钟域下的I/O

在位于不同时钟域的单周期定时循环间可能需要进行数据传输。多数FPGA终端上的数字输入与顶层FPGA终端时钟不同步，且必须进行重新同步。某些FPGA终端允许使用同步至指定时钟的外部时钟或I/O资源。如输入数据已与顶层FPGA终端时钟同步，此时可避免重新同步带来的系统开销。关于可用I/O资源同步的详细信息见指定FPGA终端的硬件文档。

在时钟域间传输数据

如要在时钟域间传输数据，可使用单周期定时循环隧道、寄存器项、握手项、局部和全局变量或存储器块FIFO。下表总结了每个方法的特性：

传输方法	FPGA资源	损耗?	并行循环间?	无限域之间?	常用于
单周期定时循环隧道	逻辑	否	否	否	数据记录
寄存器项和局部/全局变量	逻辑	是	是	是	控制，仿真
使用存储器块实现存储器项	存储器	是	是	否	数据记录
存储器块FIFO	逻辑和存储器	否	是	否	数据记录
握手项	逻辑	否	是	否	控制，仿真

单周期定时循环隧道

单周期定时循环隧道可用于在单周期定时循环上连线输入和输出信号。LabVIEW在输入信号和循环时钟域间实现握手。单周期定时循环的每个计数终止后，LabVIEW在任意输出信号和顶层FPGA终端时钟间实现握手。单周期定时循环隧道保持了LabVIEW数据流，因此不能用于在并行循环间传输数据。

寄存器项和局部/全局变量

用户可从唯一时钟域写入寄存器项或局部/全局变量。然后能够在尽可能多的用户所需的时钟域内读取寄存器项或局部/全局变量。LabVIEW将来自写入域的数据传输至读取域并使用握手机制避免数据损坏。源和目标时钟之间通过握手产生新值，写入域与读取域必须要等待几个时钟周期。由于握手机制的系统开销，从一个时钟域写入的值可能无法被目标域读取。如要避免数据写入速度高于数据读取速度，必须编程额外的逻辑以确保读取方和写入方之间的双向通信，以避免数据丢失。

注： 为确保发送时钟域的同一循环内写入的所有数据同时到达目标时钟域，请将数据放置在簇中。每个寄存器项或变量的数据均单独握手。

使用存储器块实现存储器项

仅当使用存储器块实现存储器项时，可使用终端范围或通过VI定义的存储器项存储数据，并从不同的时钟域访问数据。如要更改存储器实现，可在**存储器属性**对话框的**常规**页面选择**存储器块**用作存储器项的**实现**。在上述操作中，每个存储器项仅使用一个写入方节点和一个读取方节点。

警告 当在多时钟域使用块内存实现存储器项时, 可同时从同一地址读取和写入数据。但可能导致读取不正确的数据。

存储器块FIFO

仅当用户使用存储器块实现FIFO时，可使用终端范围的或VI定义的FIFO在时钟域间传输数据。如要更改FIFO实现，可在**FIFO属性**对话框的**常规**页面选择**存储器块**用作存储器项的**实现**。仅可通过一个时钟域写入数据至存储器块FIFO。然后仅可通过另一个时钟域读取FIFO。

注： 如使用存储器块FIFO且用于该FIFO的时钟停止，接收数据的时钟域需要一个额外的写入时钟周期，以识别最终写入的数据点。

握手项

仅可通过一个时钟域写入数据至握手项。然后可在同一时钟域或不同的时钟域读取握手项。可在任意时钟域清除握手项。LabVIEW使用握手项在写入域和读取域间实现无损耗的传输，并在读取方接收到数据时通知写入域。写入和读取时钟域之间通过握手产生新值，LabVIEW需要等待几个时钟周期。

相关概念：

- 选择FIFO实现选项
- 在单周期定时循环中执行代码
- FPGA存储器项
- 缩短FPGA VI的组合路径

- 更改顶层FPGA终端的时钟速率
- 使用流水线优化FPGA VI
- 存储和访问FPGA设计不同部分的数据

选择用作SCTL定时源的FPGA时钟

按照下列步骤选择一个FPGA时基时钟或衍生时钟作为FPGA VI中的单周期定时循环(SCTL)的定时源。SCTL默认使用FPGA终端的顶层时钟。

注： 对单周期定时循环的支持随FPGA终端变化。更多信息见FPGA终端硬件的文档。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 如所用的FPGA终端未自动添加FPGA时基时钟至项目浏览器窗口，则添加FPGA时基时钟。如要使用FPGA时基时钟作为单周期定时循环的定时源，请忽略下列步骤。
4. （可选）创建一个FPGA衍生时钟。
5. 在**项目浏览器**窗口的FPGA终端下新建一个VI或打开一个现有VI。
6. 在程序框图上放置一个定时循环。
7. 双击定时循环的**输入节点**显示配置定时循环对话框，选择下列选项之一以选择时钟：
   - **顶层定时源**－如要定时循环继承其所在项目的顶层定时源可选择此选项。如要在多个FPGA终端间重用FPGA VI可使用此选项。
   - **选择定时源**－如要使用项目中的定时源而非顶层时钟可选择此选项。然后从**可用的定时源**列表中选择一个定时源。如要使用列表以外的时钟可新建一个时基时钟，或新建一个衍生时钟。

   或者连线FPGA时钟控件至定时循环的**源名称**输入端，以编写可重用代码。

相关概念：

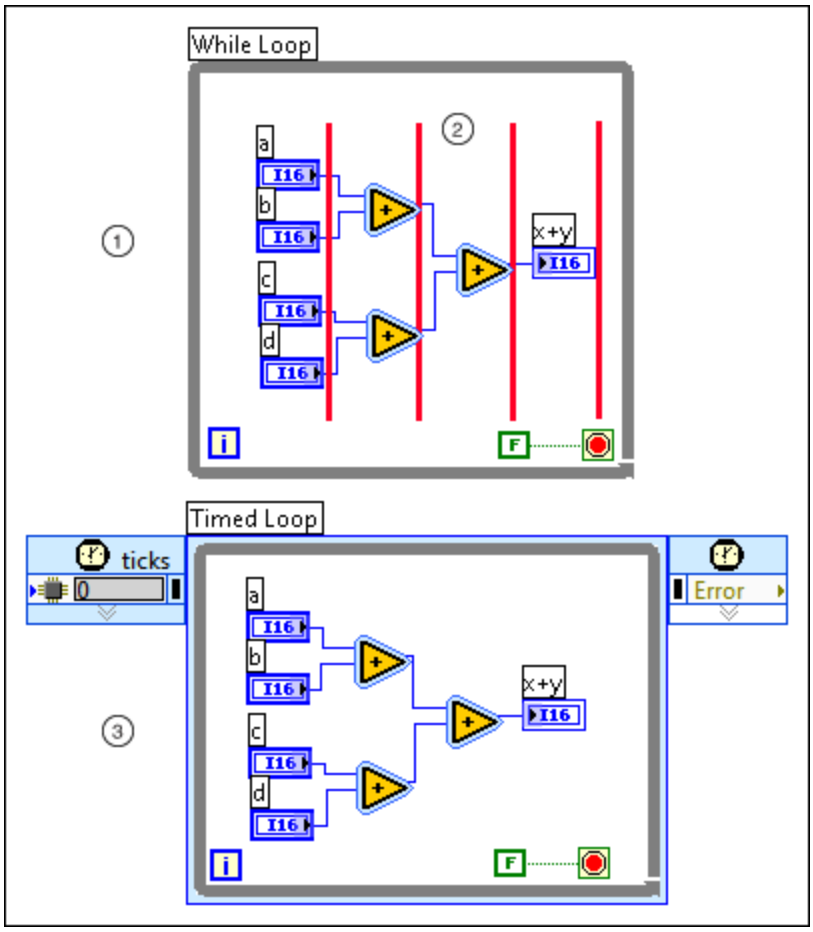
- 使用FPGA时钟和定时

- 选择用于FPGA终端的顶层时钟
- 添加FPGA终端至LabVIEW项目
- 添加FPGA时基时钟至LabVIEW项目
- 创建FPGA衍生时钟
- 添加项至项目浏览器窗口中的FPGA终端

使用单周期定时循环优化FPGA VI

LabVIEW自动优化单周期定时循环(SCTL)内的代码。与While循环内的相同代码相比，SCTL内的代码执行更快，占用FPGA终端资源更少。在FPGA终端上使用While循环时，While循环每执行一次需要占用多个时钟周期，While循环包含启用链寄存器。While循环每次执行需要占用的时钟周期的数量取决于循环内的代码。在FPGA终端上使用单周期定时循环时，单周期定时循环将在一个时钟周期内执行完循环内的所有代码。在FPGA目标上使用单周期定时循环减少了执行周期和资源占用，因为单周期定时循环不包括启用链寄存器。如果单周期定时循环包含已初始化的移位寄存器，循环第一次执行前占用一个时钟周期，以初始化移位寄存器的值。单周期定时循环类似于HDL中的定时进程。

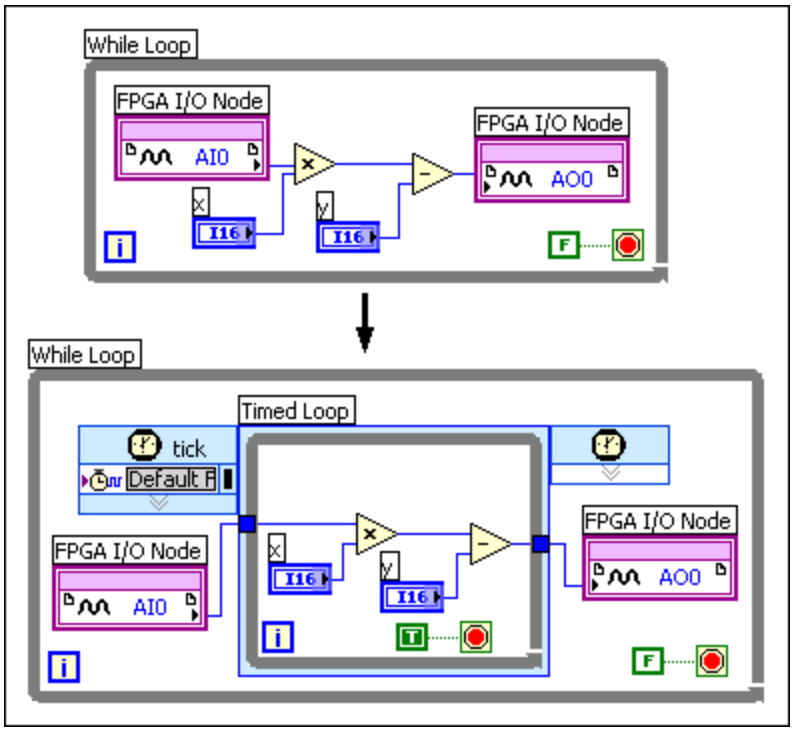
下列图示展示了执行相同代码的While循环和单周期定时循环之间的区别。



下面列出了上述程序框图的要点。

①	执行While循环内的代码需要四个时钟周期，不包括While循环占用的两个时间周期作为额外开销时间。
②	红色垂直线表示每个时钟周期内While循环执行的结束位置。
③	如时钟周期对于代码而言足够长，同样的代码放在单周期定时循环内，可以在一个时钟周期内执行完毕。

您也可使用单周期定时循环减少FPGA VI中的执行周期，以优化代码。如下图所示。



如上图所示，如果在While循环中使用单周期定时循环，则将TRUE常量连接到条件接线端，使得定时循环内的代码在While循环的每个周期执行一次。

相关概念：

- 在单周期定时循环中执行代码
- 优化FPGA VI的执行速度和大小
- FPGA VI的定时考虑因素

在SCTL中使用条件结构执行I/O

在单周期定时循环(SCTL)中可使用条件结构执行用于不同条件结构或应用状态的I/O。状态机架构以单周期定时循环的速率运行，当频率低于循环的频率时发生状态转换。

在单周期定时循环内部使用条件结构时，用于评估条件选择器的组合逻辑延时与选择器输入数据类型的宽度和条件分支数量成比例。输出隧道引入的组合逻辑延时与条件分支的数量成比例。

LabVIEW在每个时钟周期执行条件结构的所有条件分支，然后使用多路复用协议确定输出。为避免条件结构与单周期定时循环内部的数字I/O配合使用时产生未预期的行为，用户需要了解LabVIEW在单周期定时循环中同步I/O的方式。

相关概念：

- 在单周期定时循环中同步I/O

在单周期定时循环中同步I/O

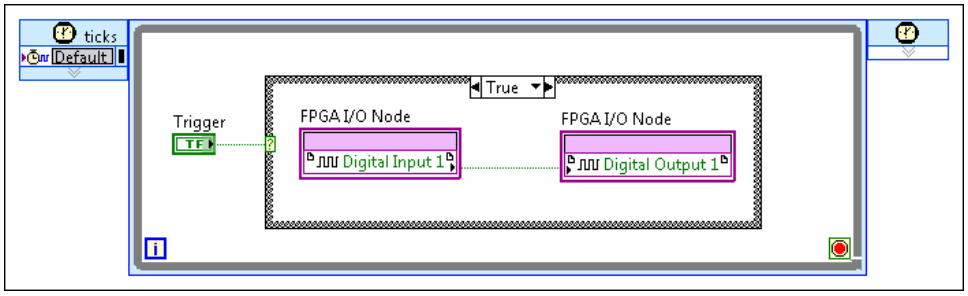
LabVIEW在FPGA硬件I/O接口和FPGA I/O节点间放置同步寄存器，该操作与终端FPGA上的操作类似。用户可指定用于输入数据的同步寄存器的数量为0、1或2。或指定用于输出数据的同步寄存器的数量为0或1。当输入被设置为**自动**（默认）时，用于输入数据的同步寄存器的数量为2，输出数据的同步寄存器的数量为1。

注： 如在单周期定时循环中指定用于数字输入和数字输出资源的同步寄存器数量为0，即在两个资源间创建了一个组合电路。组合电路可能产生输出信号毛刺并导致不可预期的动作。关于描述该场景范例代码的详细信息，见设置同步寄存器的数量为零的范例代码。

警告如指定用于嵌套在单周期定时循环内的条件结构的数字输出资源的同步寄存器数量为0，启用链置为无效时，LabVIEW不能保留数字输出资源的上一个状态，进而导致未预期的输出。NI建议在未添加用于保留上一个状态的必需逻辑的情况下，尽量避免指定同步寄存器数量为0。

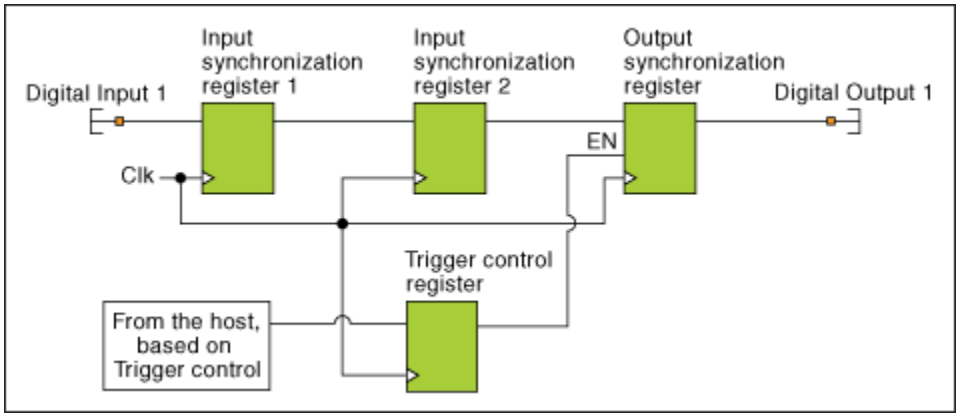
使用默认数量的同步寄存器的范例代码

假设一个带有数字I/O运算，在单周期定时循环内部运行的FPGA VI如下图所示。



数字输入信号“数字输入1”连接至数字输出信号“数字输出1”。连接至选择器接线端的触发器控制指定何时应用读取“数字输入1”的数据，并使用条件结构写入“数字输出1”。

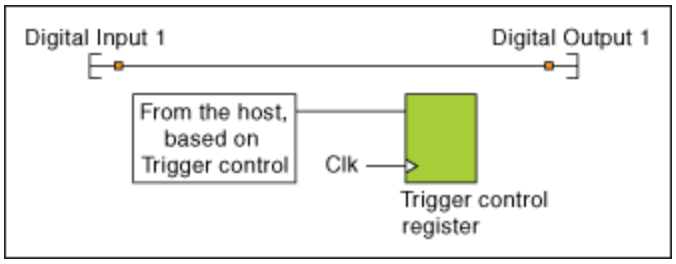
同步寄存器的数量选项被设置为默认选项**自动**时，FPGA上的电路如下列示意图所示。



在该范例中，“数字输入1”为FPGA的物理DIO引脚，通过它可直接连接两个同步寄存器。第一个同步寄存器处理亚稳态。第二个同步寄存器的输出直接连至“数字输出1”的对应同步寄存器，该寄存器连至FPGA上的输出DIO引脚。触发器控制寄存器输出连至输出同步寄存器的启用链输入端，以判定何时输出信号。

设置同步寄存器数量为0的范例代码

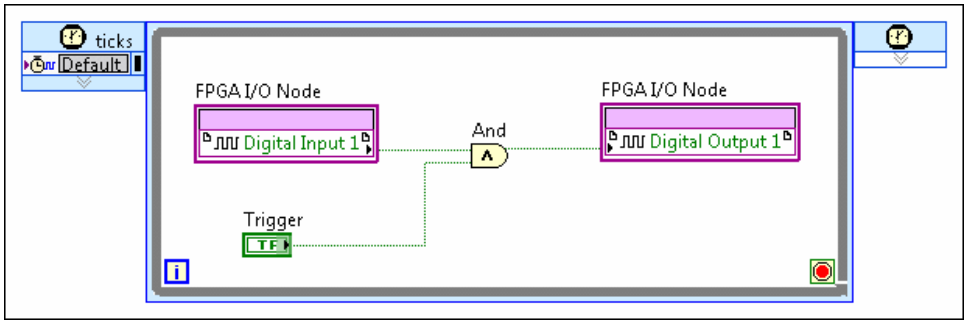
假设程序框图与上图一致，但用于数字输入和输出的同步寄存器的数量被设置为0。FPGA的等效电路如下所示。



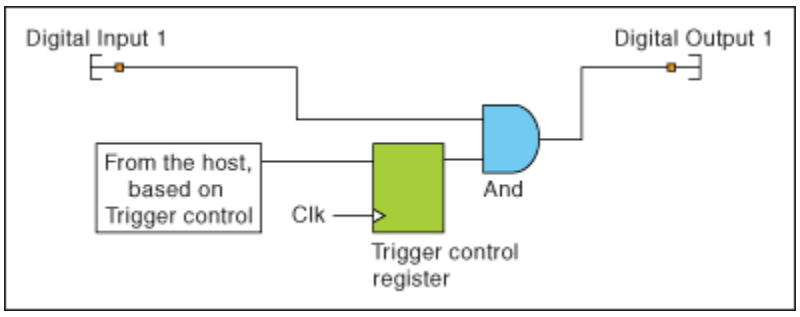
此时数字输入信号"Digital Input 1"直接通过硬接线连至FPGA上的数字输出信号"Digital Output 1"。编译进程不包含任何用于判断何时读取“数字输出1”和写入“数字输出1”的控制逻辑。在上述情况下，比特位文件下载至FPGA后，数字I/O立即连接彼此。

使用0个同步寄存器和一个逻辑函数的范例代码

如在单周期定时循环内，必须设置I/O同步寄存器的数量为0且对于“设置同步寄存器数量为0的范例代码”中描述的动作不满意，用户必须确保应用程序在输入和输出间包含逻辑。通过在I/O输入和输出间添加逻辑可避免使用独立于控制逻辑之外的直接连线。下列程序框图显示了一个不带有同步寄存器的执行代码选项。



上述程序框图使用"逻辑与"函数替代条件结构，控制何时信号到达“数字输出1”。“数字输入1”节点和触发器控制为“逻辑与”函数的输入，“逻辑与”函数的输出连接至“数字输出1”节点。FPGA的等效电路如下所示。



上述电路与“使用默认数量的同步寄存器的范例代码”中的电路类似。仅当触发器控制设置为TRUE时，“数字输入1”和“数字输出1”间才建立连接。但电路之间的区别在于：“数字输入1”和“数字输出1”间的信号路径不包含寄存器。“数字输出1”可能独立于控制单周期定时循环的时钟变化。

注： 用于实现“与”函数逻辑的组合逻辑可能引起输出端信号毛刺，并导致未预期的动作。

相关概念：

- 在SCTL中使用条件结构执行I/O
- FPGA VI的数据流和启用链

配置布尔控件的机械动作

布尔控件的机械动作用于创建与真实仪器（如示波器和万用表等）类似的前面板行为。右键单击布尔控件，可从**机械动作**菜单中选择不同动作。

FPGA VI以硬件模式运行而不是以仿真模式运行时，FPGA VI的布尔控件的机械动作不同。以仿真模式运行时，FPGA VI的布尔控件的机械动作与在LabVIEW中相同。在FPGA终端上，使用**单击时转换**、**释放时转换**或**保持转换直到释放**作为输入控件，在主VI向FPGA终端写入新值时改变当前值，并在主VI再次写入新值前保持该新值。使用**单击时触发**、**释放时触发**或**保持触发直到释放**作为输入控件，仅在主VI向FPGA终端写入新值时改变当前值，并在FPGA终端读取新值后恢复至先前值。

判定在FPGA设计中使用何种数据类型

下表介绍了具体情况下使用整型、定点和单精度浮点型数据的含义。通过下表确定最适合于用户FPGA应用的数据类型。

使用场景	数据类型	数据精度	FPGA资源使用	延迟
需要在已校准定点或未校准整数I/O节点输出间选择。	整型	与字节长度成比例。较高的动态范围需要较长的字节长度。	与字节长度成比例。	与时钟速率成比例。
设计包含将多个整型数据打包为32位或64位字节。				
设计需要执行位运算。例如，掩码或反转。				
设计需要资源有效的算术。	定点			
需要从cRIO机箱采集模拟I/O。				
需要使用高吞吐率数学函数。				
在同一数据路径中需要表示非常大和非常小的数值。例如，累加器。	单精度浮点型	即使对于高动态范围数据路径，仍保持24位精度。	尤其对于类似“加”、“减”和“乘”的函数，显著高于定点型。	相对于使用定点型或整型数据类型的运算，需要更多的时钟周期。
需要快速成型。使用单精度浮点型快速获取功能性硬件设计。必要时，可转换为定点以优化FPGA性能或资源使用量。				

在FPGA VI中支持单精度数据类型的函数

下列函数用于FPGA VI时，支持单精度浮点数据类型。

数值函数

- 绝对值
- 添加
- 复合运算
- 减1

- 除
- 加1
- 计算机Epsilon
- 乘
- 取相反数
- 负无穷大
- 正无穷大
- 倒数
- 按2的幂缩放
- SGL数值常量
- 符号
- 平方
- 平方根
- 减

数学与科学常量

- 以10为底的e的对数
- 自然对数的底数
- Pi的自然对数
- 2的自然对数
- 10的自然对数
- Pi
- Pi除以2
- Pi乘以2
- e的倒数
- Pi的倒数

转换函数

- 转换为单字节整型
- 转换为定点
- 转换为长整型

- 转换为64位整型
- 转换为单精度浮点数
- 转换为无符号单字节整型
- 转换为无符号长整型
- 转换为无符号64位整型
- 转换为无符号双字节整型
- 转换为双字节整型

比较函数

- 等于?
- 等于0?
- 大于?
- 大于0?
- 大于等于?
- 大于等于0?
- 判定范围并强制转换
- 小于?
- 小于0?
- 小于等于?
- 小于等于0?
- 最大值与最小值
- 不等于?
- 不等于0?
- 选择
- 非法数字/路径/引用句柄?

通道线端点

无

串流

- 写入

- 带中止功能的写入
- 读取
- 带中止功能的读取

标记

- 写入
- 带中止功能的写入
- 读取
- 带中止功能的读取

累加器Tag

- 写入
- 带中止功能的写入
- 读取
- 带中止功能的读取

有损串流

- 写入
- 带中止功能的写入
- 读取
- 带中止功能的读取

单元素串流

- 写入
- 带中止功能的写入
- 读取
- 带中止功能的读取

在FPGA应用程序中使用子VI

LabVIEW允许用户将常用的代码封装为子VI，以方便其在程序框图中的重用。开发FPGA应用时，在下列场景使用子VI：

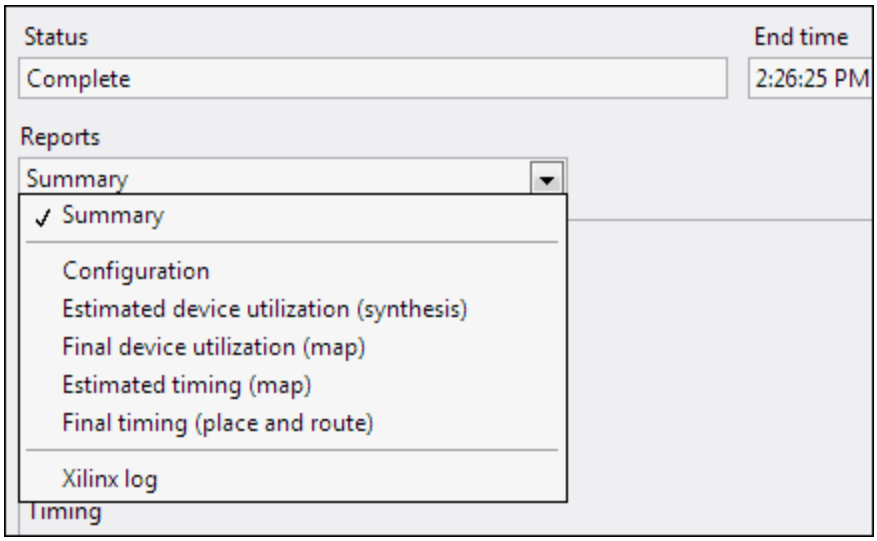
- **使用多个FPGA VI**—LabVIEW仅能运行一个顶层FPGA VI，但将多个FPGA VI调用为顶层VI的子VI，即可实现运行多个FPGA VI。
- **节省FPGA资源**—子VI中的前面板对象不直接与主控VI通信，从而节省了额外的FPGA资源。
- **在多个项目中重用代码**—创建可重用的模块化代码能简化代码审查和更新过程，避免代码重复，从而有助于高效地组织、管理、测试和调试应用程序。

判定何时使用重入或非重入子VI

配置子VI为重入或非重入。默认情况下，在FPGA终端下创建的VI为重入子VI。下表为指定使用条件下的应用建议。

子VI类型	设计
重入	子VI不访问共享资源（例如，I/O）的时候使用。
非重入	子VI访问共享资源时使用。
非重入	多个子VI的实例在VI中共享数据时使用。例如，功能性全局变量。

对于其他设计执行，可使用“编译状态”窗口**报告**下拉菜单中的预估设备使用量和定时报告验证满足FPGA应用程序需求的子VI配置。编译FPGA VI后将显示**编译状态**窗口。用户可能需要反复几次，才能找到最适合应用的子VI类型。



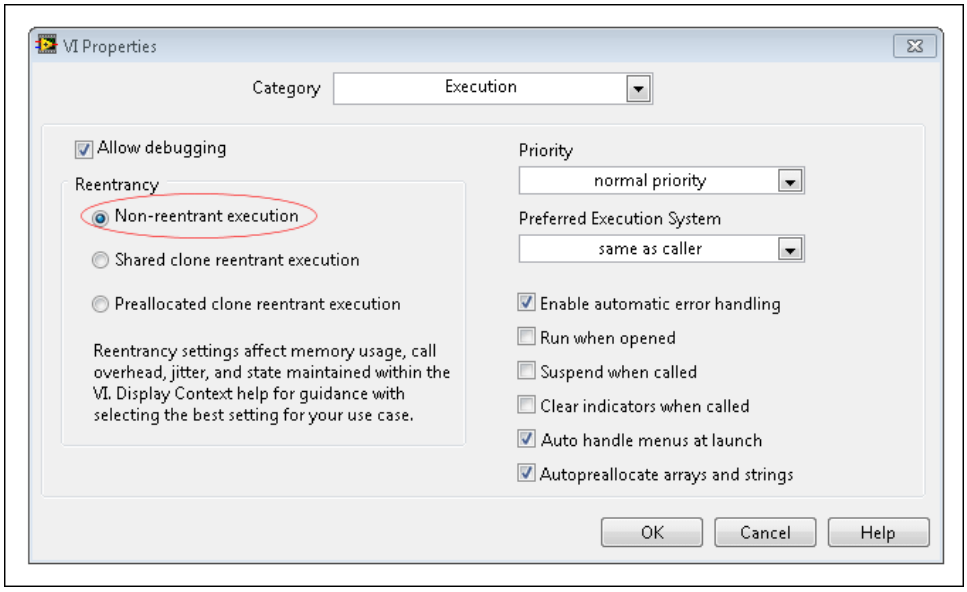
相关概念：

- FPGA存储器项
- 管理共享的资源
- 优化FPGA VI的执行速度和大小
- 存储和访问FPGA设计不同部分的数据
- 教程：创建测试台

设置子VI为非重入

按照下列步骤设置子VI为非重入。

1. 按下<Ctrl-I>显示**VI属性**对话框。
2. 在**类别**下拉菜单中选择**执行**。
3. 在**重入**部分选择**非重入执行**。
4. 单击**确定**。



在子VI中使用I/O、时钟、寄存器项、存储器项、FIFO和握手项

使用名称控件创建能够被不同I/O、时钟、寄存器、存储器、FIFO或握手资源重用的子VI。仅可在重入FPGA子VI中使用名称控件。

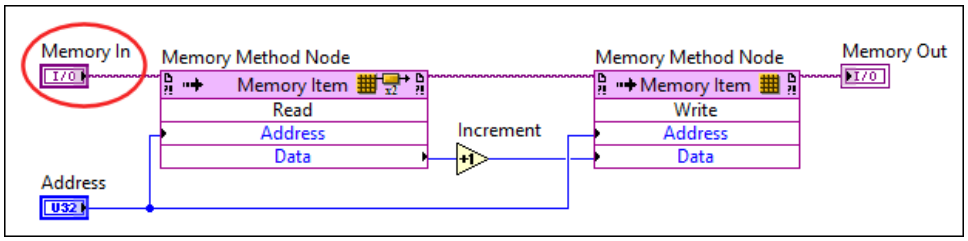
注： FPGA名称控件位于**名称控件**选板。FPGA模块还包含用于每个FPGA名称控件的程序框图常量。

按照下列步骤启用子VI，接收FPGA I/O、时钟、寄存器、存储器或FIFO资源。

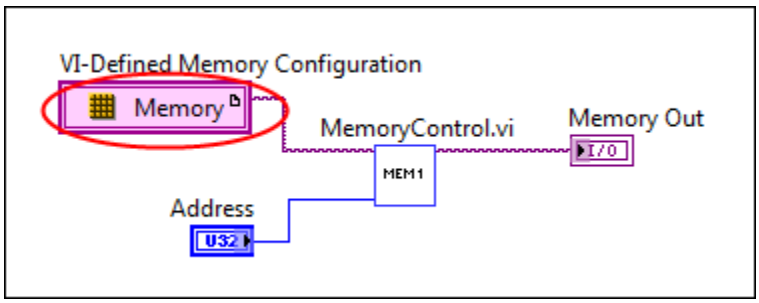
1. 在子VI的程序框图内放置FPGA I/O节点、寄存器方法节点、存储器方法节点、FIFO方法节点或握手方法节点。如要选择FPGA时钟，在子VI的程序框图上放置一个单周期定时循环。
2. 右键单击**FPGA I/O输入**、**寄存器输入**、**存储器输入**、**FIFO输入**、**握手输入**或**源名称**输入端并选择**创建»输入控件**。
3. 分配I/O、寄存器、存储器、FIFO、握手或时钟控件至子VI的连线板接线端。

传输I/O、时钟、寄存器项、存储器项、FIFO或握手项至子VI。

下列程序框图为MemoryControl.vi，其中**存储器输入**连线至存储器方法节点。



在下列程序框图中，MemoryControl.vi用作引用子VI。注意，通过VI定义存储器配置节点指定下列框图中调用VI使用的存储器项。该设计阐明如用户需要使用其他存储器项执行MemoryControl.vi，仅需使用“通过VI定义存储器配置”节点更改**存储器输入**连接的项。



FPGA名称控件的限制条件

下列限制条件适用于FPGA I/O、cRIO I/O、存储器、FIFO、握手和时钟控件：

- 仅当编程FPGA VI时，可更改FPGA名称控件的值。FPGA VI在开发计算机上运行或使用交互式前面板通信时，不能更改控件的值。
- 仅当在FPGA终端下编程VI时，FPGA名称控件可用。编程主控VI时，FPGA名称控件不可用。
- FPGA名称控件可与其他名称控件、事件发生引用句柄和其他数据类型被捆绑至簇。但不能在顶层FPGA VI的前面板中使用上述簇。否则，编译FPGA VI时LabVIEW会报错。

相关概念：

- FPGA存储器项
- 存储和访问FPGA设计不同部分的数据

配置FPGA I/O节点名称控件

FPGA I/O名称控件用于创建带有可配置I/O项的重入子VI。FPGA I/O控件用作子VI的输入时，输入端仅能够是支持用于FPGA I/O输入控件的方法和属性的扩展集的I/O项。所有启用的方法必须按名称匹配，且其参数必须按名称、顺序和数据类型匹配。所有启用的属性必须按名称和数据类型匹配。

按照下列步骤使用**配置FPGA I/O名称控件类型**对话框配置FPGA I/O名称控件。

1. 右键单击FPGA I/O控件或常量，从快捷菜单中选择**配置I/O类型**显示**配置FPGA I/O名称控件类型**。
2. 从**I/O项**列表中选择一个FPGA I/O项，该项支持要用于FPGA I/O控件的所有方法和属性。所选I/O项的方法和属性将出现在**I/O项类型**列表中。
3. 单击**替换全部**复制**I/O项类型**列表中的全部方法和属性至**I/O名称控件类型**列表。不能移动单个I/O项类型至**I/O名称控件类型**。
4. 单击**移除**从**I/O名称控件类型**列表中移除无需由I/O项传递至控件的方法和属性。如要控件具有广泛的可重用性，应移除不想用于FPGA I/O输入控件的某个特定资源的方法和属性。单击**I/O名称控件类型**列表中的项，可在**I/O名称控件类型详细信息**部分查看项的信息。
5. 单击**确定**，关闭对话框。

重用FPGA对象

可在多个FPGA终端间复制、剪切或粘贴FPGA VI，以创建新的FPGA VI终端特定应用程序实例。可使用“项目浏览器”窗口在多个FPGA终端间复制、剪切和粘贴FPGA I/O项、FPGA时钟、寄存器项、存储器项、FIFO和握手项。该项的副本位于**项目浏览器**窗口中选中的FPGA终端下。FPGA项的支持随FPGA终端变化。

提示 也可以在“项目浏览器”窗口拖放多个FPGA VI和FPGA项，以在多个FPGA终端下创建副本。

也可以创建其他的FPGA VI实例及在LabVIEW项目间复制FPGA项。打开待执行复制

操作的项目，然后在项目间复制或拖曳FPGA项。

重用FPGA VI

只要FPGA终端支持全部用于FPGA VI的FPGA I/O项、FPGA时钟或FPGA FIFO，同一个FPGA VI可用于位于相同类或不同类的多个FPGA终端下。由于FPGA VI特性随FPGA终端变化，所以仅可在某些FPGA终端上编译和运行FPGA VI实例。复制使用FPGA I/O项、时钟或FIFO的FPGA VI至FPGA终端时，需将附加项与FPGA VI一并复制。否则，FPGA VI的**运行**按钮将显示为断开，无法编译和运行VI。

注： 创建FPGA VI实例时，LabVIEW以一个用于每个FPGA VI实例的磁盘文件为参考。如修改了某个FPGA VI的应用实例，将出现“同步其他应用程序实例”工具栏按钮。单击**同步其他应用程序实例**按钮，将改动应用至所有FPGA VI实例。可使用**另存为**创建FPGA VI的副本，而不会影响磁盘上相同的文件。

重用FPGA I/O项

FPGA I/O项可在多个FPGA终端间被复制。如新的FPGA终端不支持为FPGA I/O项配置的I/O资源或该I/O资源已被占用，该项在“项目浏览器”窗口中显示为红圈内包含一个白色的!。同样，如新FPGA终端下的FPGA VI在程序框图上包含FPGA I/O节点，且使用不支持的FPGA I/O项，**运行**按钮显示为断开，且无法编译或运行VI。如FPGA I/O项断开，右键单击**项目浏览器**窗口的FPGA I/O项，从快捷菜单中选择**选择资源**，显示“选择资源”对话框。从**选择资源**对话框中选择FPGA终端支持的I/O资源，然后单击**确定**按钮。也可以从**项目浏览器**窗口右键单击FPGA I/O项，从快捷菜单中选择**移除**，从项目中移除该项。必须重新配置FPGA I/O节点。

重用FPGA时钟

可在多个FPGA终端间复制FPGA时钟。如新的FPGA终端不支持用户配置FPGA时钟的时钟资源或该时钟资源已被占用，FPGA时钟在“项目浏览器”窗口中显示为红圈内包含一个白色的!。同理，如新FPGA终端下的FPGA VI在程序框图上包含单周期定时循环，且使用不支持的FPGA时钟，尝试编译和运行VI时将弹出“代码生成错误”窗口

并显示错误信息。如FPGA时钟断开，右键单击**项目浏览器**窗口的FPGA时钟，从快捷菜单中选择**属性**，显示“FPGA时基时钟属性”对话框或“FPGA衍生时钟属性”对话框。如复制FPGA时基时钟，必须从**FPGA时基时钟属性**对话框的**资源**下拉菜单中选择FPGA终端支持的时钟源。复制FPGA衍生时钟时，通过配置FPGA衍生时钟可选择FPGA终端支持的时钟配置。也可以从**项目浏览器**窗口右键单击FPGA时钟，从快捷菜单中选择**删除**，从项目中删除FPGA时钟。然后必须使用新建FPGA时钟重配置单周期定时循环。

重用寄存器项

寄存器项可在多个FPGA终端间被复制。

重用FPGA存储器项

存储器项可在多个FPGA终端间被复制。如新FPGA终端不支持复制的存储器项，编译FPGA VI时“代码生成错误”窗口将报告编译失败。可以右键单击“项目浏览器”窗口中的存储器项，从快捷菜单中选择**属性**，显示“存储器属性”对话框。然后配置存储器项并重新编译FPGA VI。也可以从**项目浏览器**窗口右键单击存储器项，从快捷菜单中选择**从项目中删除**，从项目中删除存储器项。

重用FPGA FIFO

FPGA FIFO可在多个FPGA终端间被复制。如新FPGA终端不支持复制的FPGA FIFO，编译FPGA VI时“代码生成错误”或“编译状态”窗口将返回编译失败的报告。可以右键单击“项目浏览器”窗口中的FPGA FIFO，从快捷菜单中选择**属性**，显示相应的“FIFO属性”对话框。然后可配置FPGA FIFO并重新编译FPGA VI。也可以在**项目浏览器**窗口右键单击FPGA FIFO，从快捷菜单中选择**从项目中删除**，从项目中删除FPGA FIFO。

重用握手项

握手项可在多个FPGA终端间被复制：

- **通过VI定义的握手项**：使用通过VI定义的握手项创建重入子VI并避免资源冲突。在重入FPGA VI中配置通过VI定义的握手项，LabVIEW将为用于VI的每个实例的握

手项创建独立的副本。
- **终端范围的握手项**：如需握手项可见且可通过**项目浏览器**窗口配置，请使用终端范围的握手项。终端范围的握手项在**项目浏览器**窗口中同一终端下的任意FPGA VI内均可用。

重用CLIP项

同一CLIP可在不同的FPGA终端间使用。但不能在“项目浏览器”窗口下的终端间拖曳CLIP项。必须添加CLIP至新建终端。情况允许时，用户还需要按照在原始终端关联I/O项的方式，配置与CLIP关联的I/O项。

相关概念：

- 在项目浏览器窗口管理FPGA应用

创建FPGA VI时使用LabVIEW类

创建FPGA VI时可使用LabVIEW面向对象编程技术部分内容。

支持的LabVIEW类功能

- 类常量、输入控件和显示控件
- 类方法
- 在类内使用全部FPGA支持的数据类型
- 其他类的私有数据包含的类

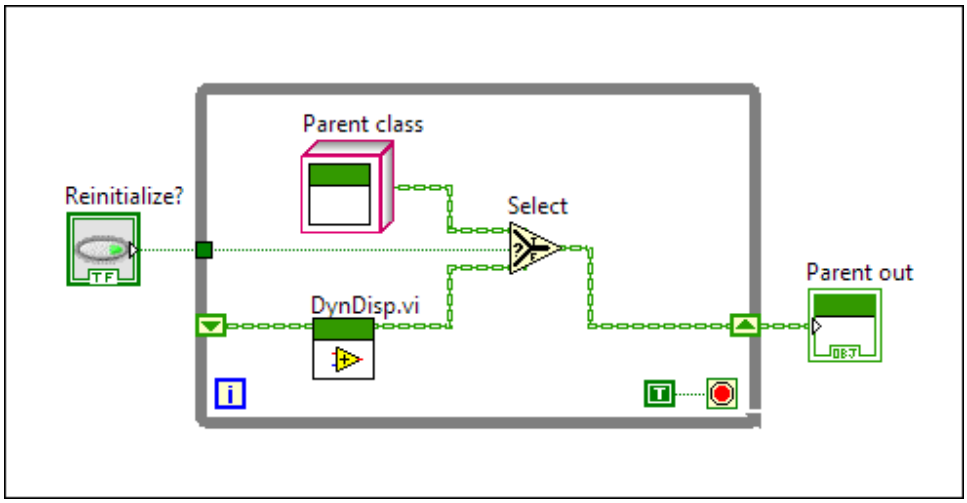
> **注：** 由于LabVIEW必须能够在编译时转换全部类，因此不能在顶层FPGA VI中使用LabVIEW类。

继承和类的编译时间精度规范

在LabVIEW面向对象编程中，运行时对象的类型可与连线类型一致或为其子孙类。编译FPGA VI至位文件时，数据连线为固定的。即编译器不能动态调用多个代码的

实现。该限制意味着编译器必须能够静态判定运行时连线的类型。创建FPGA VI时使用下列类型规范：

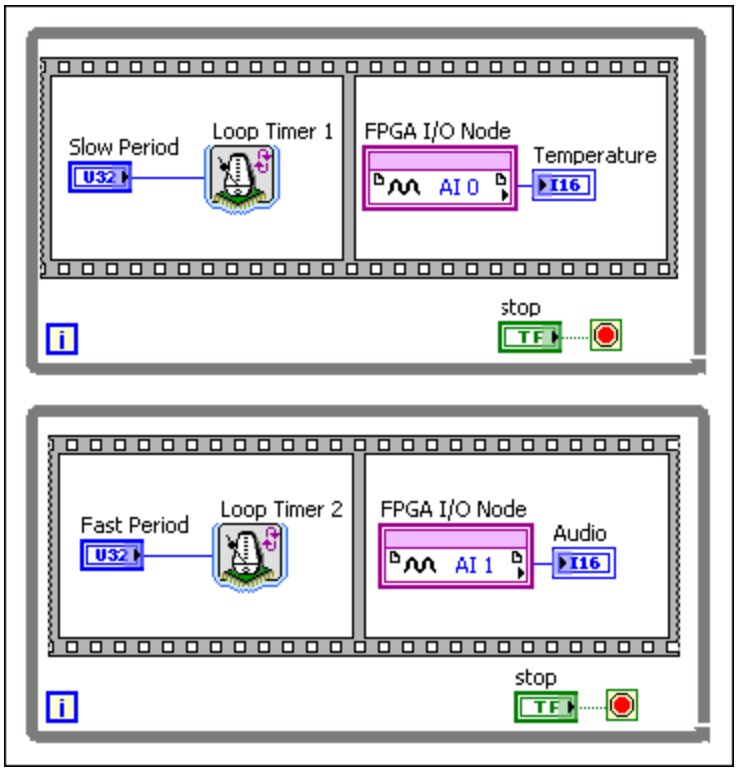
- 连线子类接线端至父类接线端。
- 与其他子VI调用相比，由于重写调用是在编译时确定的，因此FPGA VI不会产生动态分配的系统开销。
- 如FPGA VI带有一个LabVIEW类输入且非重入，用户可在VI层次结构中的多个位置调用VI，并在不同的位置使用不同的连线类型。但当编译器分析VI层次结构时，连线类型必须处理全部位置的同一个类。这是用于静态分配和动态分配VI。
- 如程序中包括类的局部变量，请确保全部写入与类的类型匹配。如下所述：
  - 对于LabVIEW类输入控件及其相关的写局部变量，请确保全部连线至写局部变量的数据为同一类的类型。如控件位于连线板，必须确保连线至控件对应接线端的数据线为同一类的类型。
  - 对于LabVIEW类显示控件及其相关的写局部变量，请确保全部连线至写局部变量和显示控件的数据线为同一类的类型。
- 如程序中包括类的局部变量且某些变量为读取方，请确保全部写入与默认的类的类型匹配。如下所述：
  - 如程序包含与输入控件关联的读取局部变量或与已连线的输入控件关联的局部变量，必须确保全部连接至写局部变量的数据线为默认类型。如输入控件位于连线板且已连线控件对应的接线端时除外。
  - 如程序包含与显示控件相关的读取局部变量，必须确保全部连线至显示控件和写局部控件的数据线为默认的类型。
- 数组中的全部对象必须处理同一类。
- 所有帧的条件结构隧道必须处理同一类。
- 编译器在初始化接线端处理用于移位寄存器和反馈节点的类，并在后续循环内使用该类型。未初始化的移位寄存器和反馈节点处理连线的类。如在循环内引入子类，编译将失败。在下图中，FPGA VI仅当`DynDisp.vi`总是在输出端返回与输入端相同的类型时才能执行编译。



使用并行操作

相对于基于处理器的终端，FPGA上的并行操作通常会加快执行速率并降低抖动。每个并行操作在其专有的FPGA硬件区域执行，以实现真正的并发执行。因此，并行FPGA操作的整体执行时间等于较慢的操作的执行时间。单处理器上相同操作的整体执行时间等于全部执行时间的和值。

如要创建并行操作可在单个程序框图上适用多个独立的While循环。例如，实现多个数据采集引擎（每个均适用独立的采样速率）。如下列程序框图所示。



在包含高频和低频信号的系统中使用独立的采样速率可高效获取数据。配置一个数据采集引擎为高采样率，以测量高频信号。配置一个数据采集引擎为低采样率，以测量低频信号。

如在并行操作间使用共享资源可能遇到禁用并行执行的风险，因为每个操作均需等待，直至共享资源可用才能执行。可能的共享资源包括数字I/O资源、模拟I/O资源、存储器项、寄存器项、中断线、局部和全局变量及非重入子VI。

提示 每个并行操作使用一定数量的FPGA空间。如FPGA上空间不足且具有相同的并行操作，通过为操作创建子VI可节省空间并使其为非可重用。但为操作创建非可重用子VI即牺牲了并行执行。

通过连线在循环间传输数据将引入数据流延迟，进而影响循环并行执行。

- 存储器项
- 寄存器项
- 全局或局部变量
- FIFO

相关概念：

- 优化FPGA VI的执行速度和大小
- 管理共享的资源
- 使用FPGA I/O

FPGA VI对可变大小数组的支持

用户可创建数组大小可变的FPGA VI，该数组在编译时将转换为单个大小。编译FPGA VI至位文件时，代码的多个实现不能被动态调用。该限制表明LabVIEW必须能够在编译时判定数组输入或输出的大小。下表介绍了在FPGA应用中实现可变大小数组的操作、说明和限制条件。

数组运算	对于在FPGA模块中支持数组输入或输出的函数，LabVIEW通过检查函数的输入判定数组输出的大小。如函数仅包含标量或固定大小的数组输入，LabVIEW计算固定大小的数组输出。如其不能计算固定的大小，LabVIEW将返回错误。
类和簇	编译时可转换为单个大小的数组可被嵌入至簇和类中。
子VI	LabVIEW跟踪记录子VI数组的大小，以将其传入或传出FPGA VI。对于非重入子VI，如两个调用得到的数组大小不匹配，LabVIEW将对该前面板控件和全部对该VI的调用返回错误。
移位寄存器	未初始化的移位寄存器默认数组大小为零。对于初始化移位寄存器，LabVIEW通过检查输入接线端计算数组大小。如初始化输入端的大小不匹配，LabVIEW将返回错误。
反馈节点	LabVIEW比较“反馈节点”输入端与初始化接线端的数组大小。如大小不匹配，LabVIEW将返回错误。对于未初始化的反馈节点，LabVIEW为初始化接线端使用零值。
选择函数	仅当两个输入数组大小编译时静态地转换为同一固定大小时，LabVIEW才能推断**选择**函数的输出数组大小。否则，LabVIEW可返回错误。
条件结构	仅当“条件”结构中全部程序框图的输入数组大小编译时静态地转换为同一固定大小，LabVIEW计算“条件”结构输出隧道的数组大小。否则，LabVIEW可返回错误。
For循环	LabVIEW根据For循环执行的次数，计算自动索引输出隧道的数组大小。必须连线常量值至计数接线端。如不能静态转换数组输出的大小，LabVIEW将返回错误。
强制转换	如果将可变大小的数组连接到不匹配的固定大小接线端（如子VI接线端和显示控件），则会根据固定大小接线端的长度强制填充或截断数据。如果将固定大小的数组

	连接到可变大小的接线端，将强制可变大小的接线端使用固定大小数组的大小。
捆绑数组	LabVIEW计算捆绑数组的值等于或小于捆绑值。如捆绑数组编译时不能转换为单个大小，LabVIEW将返回错误。
自动预分配数组和字符串	如在**VI属性**对话框的**执行**页中勾选**自动预分配数组和字符串**复选框，可混合使用固定大小和可变大小的数组，LabVIEW将强制转换数据。

从数组函数返回编译时可转换的数组

在FPGA VI中，仅可使用编译时可将数组转换为单个大小的数组函数。即某些数组属性（例如，用户读取/写入元素的长度或索引）必须为常量值。下列两种方式可选择其一：直接连线常量值至函数，或依赖常量折叠传输值。

注： 尽可能使用常量输入，因为其在FPGA VI中比较高效。

下列数组函数编辑时能够返回可转换为单个大小的数组。

函数	**所需常量**
数组子集	**索引**和**长度**输入必须为常量。也可将所有输入端设置为常量。
创建数组	可使用常量或非常量输入。
簇至数组转换	可使用常量或非常量输入。
抽取一维数组	可使用常量或非常量输入。
删除数组元素	**长度**和**索引**输入必须为常量。也可将所有输入端设置为常量。
初始化数组	**元素**输入端必须为非常量。**维度大小**输入端必须为常量。也可将所有输入端设置为常量。
数组插入	**索引**输入端必须为常量。也可将所有输入端设置为常量。
交织一维数组	可使用常量或非常量输入。
替换数组子集	可使用常量或非常量输入。

函数	所需常量
重排数组维数	**维度大小**输入端必须为常量。
反转一维数组	可使用常量或非常量输入。
一维数组循环移位	可使用常量或非常量输入。
拆分一维数组	**索引**输入端必须为常量。也可将所有输入端设置为常量。

在FPGA VI中使用固定大小的数组

FPGA模块支持编译时可转换为单个大小的一维数组。如LabVIEW不能计算数组的单个大小，请考虑在FPGA VI中使用固定大小的数组。

创建固定大小的数组输入控件

按照下列步骤创建固定大小的数组输入控件：

1. 添加数组至前面板窗口。数组由索引框、元素框和可选标签组成。索引框位于左侧，元素框位于右侧。
2. 如**控件**选板不可见，右键单击元素显示框或前面板窗口，打开**控件**选板。
3. 拖曳输入控件或显示控件至数组。例如，拖曳一个数值输入控件至数组。
4. 右键单击数组输入控件的索引，从快捷菜单中选择**设置大小**。
5. 在“属性”对话框的“大小”页面，选择**固定**。
6. 输入数组中的元素数量。
7. 单击**OK**按钮。

创建固定大小的数组常量

按照下列步骤创建固定大小的数组常量：

1. 在程序框图上添加一个数组常量。数组常量带有一个索引框、一个空的元素框和一个可选标签。索引框位于左侧，元素框位于右侧。
2. 添加一个常量至数组。

3. 右键单击数组常量的索引，从快捷菜单中选择**设置大小**。
4. 在“属性”对话框的“大小”页面，选择**固定**。
5. 输入数组中的元素数量。
6. 单击**OK**按钮。

使用定点数据类型

定点数据类型提供了浮点型数据类型的部分灵活性，但仍保留了整数算术的大小和运算速度优势。关于定点数据类型和使用该数据类型的详细信息见ni.com。

注： 并非所有FPGA模块VI和函数均支持定点数据类型。此外，某些函数仅提供有限的定点支持。例如，正弦发生器Express VI仅在输入接线端支持定点数。

警告 如连线定点数至整数接线端，可能会丢失小数部分。

定点配置对FPGA资源的影响

在FPGA VI中使用定点数时，为特定函数选择正确的溢出和四舍五入模式非常重要。高吞吐率数学函数和部分数值函数带有配置对话框，可用于模式选择。如能够产生溢出或四舍五入，这些模式将影响FPGA VI所需的FPGA资源量。如不能产生溢出或四舍五入，操作不需要使用额外的FPGA资源。

多数情况下，在函数的配置对话框中勾选**匹配至源**复选框可避免溢出。如勾选此复选框，LabVIEW将尝试调整输出数据类型的宽度和范围，从而不会产生溢出或四舍五入。但此调整并不适用于所有情况。例如，函数包含除法时会一直产生四舍五入。且某些函数不包含此复选框。

溢出模式的影响

通常溢出模式以下列几种方式影响FPGA资源：

- **饱和**－需要FPGA资源判定输入值是否位于输出类型的范围内并选择返回原始值或饱和值。
- **截位**－所需的FPGA资源少于**饱和**模式。

四舍五入模式的影响

通常四舍五入模式以下列几种方式影响FPGA资源：

- **截断舍入**－移除位，因此不需要占用FPGA资源。但此模式对于多数数据流将产生最大的平均误差。此模式为整型操作的默认模式。
- **半值向上（非对称）**－添加至输出数据类型的最低有效位。此选项需要一个宽度等于输出数据类型的加法器。
- **奇偶舍入**－所需FPGA资源最多，且在三种舍入模式中生成最长的组合路径。但此模式对于多数数据流可返回统计意义上最正确的结果，因此其为定点数据类型的默认四舍五入模式。

注： “高吞吐率除”和“高吞吐率倒数”函数使用输出值向零取整的四舍五入模式。此模式将输出值四舍五入为数据类型能够表示的最接近的值。如值为整数，LabVIEW进行截断取整。如值为负数，假设至少有一位删除的位不为零，LabVIEW删除最低有效位(LSB)并为剩余的LSB添加符号位。例如，输出值x位于连续值s和t之间（$s < x < t$），如x为正数，LabVIEW设置x等于s。如x为负数，LabVIEW设置x等于t。对于这两个函数，与其它四舍五入模式相比，向零取整使用的FPGA资源最少。不能为这些函数指定不同的四舍五入模式。

使用高吞吐率数学函数

使用高吞吐率数学函数执行高吞吐率数学并分析FPGA终端上的定点数。上述函数与数值函数类似，但支持较高的吞吐率、单周期定时循环内的握手接线端、输入/输出寄存器和自动流水线。

“高吞吐率数学”函数和“数值”函数的区别

“高吞吐率数学”函数与LabVIEW“数值”函数在下列方面不同：

- **附加函数**－高吞吐率数学函数包含支持定点数据类型的三角函数、对数函数和直角/极坐标转换函数。
- **受限的数据类型支持**－相对于数值函数，高吞吐率数学和基本元素函数支持的数据类型较少。关于支持的数据类型的详细信息，分别见高吞吐率数学或基本元素主题。
- **支持通用单周期定时循环**－在单周期定时循环内不能放置某些“数值”函数。无论函数占用几个执行时间周期，均可在单周期定时循环内放置全部“高吞吐率数学”函数。在单周期定时循环内，上述函数可能显示握手接线端。通过上述接线端确保算法仅使用有效的数据操作。高吞吐率数学函数还能够控制组合路径的长度，以改进函数可编译的时钟速率。
- **支持较高的吞吐率**－如需要在单周期定时循环内流水线“数值”函数，必须手动流水线上述函数。但多数“高吞吐率数学”函数均带有一个**吞吐率**控件。LabVIEW自动流水线函数，以获取指定的吞吐率。
- **标签接线端**－“高吞吐率数学”函数可配置为显示数值接线端的编码、字节长度和整数字节长度。通过即时帮助窗口可查看函数或接线的配置信息。

除非需要使用“高吞吐率数学”函数的特有优势，否则建议使用LabVIEW“数值”函数。创建VI时，“数值”函数更易于使用且比“高吞吐率数学”函数的适用平台更广。例如，“高吞吐率数学”函数不支持NI RT终端。

配置函数

如要配置“高吞吐率数学”函数，可双击函数或右键单击函数，从快捷菜单中选择**配置**。LabVIEW将显示函数的配置对话框。该对话框可用于配置函数执行方式和返回结果的各个方面。通过配置对话框设置输入接线端的编码、字节长度和整数字节长度，以避免不支持的定点数问题。

配置函数外观

每个高吞吐率数学函数可以展开或折叠的方式显示。展开视图可显示每个接线端的编码、字节长度和整数字节长度。折叠视图可节省程序框图的空间。如要配置函数的外观，右键单击函数并选择**展开视图**或**折叠视图**。

相关概念：

- 在单周期定时循环内放置高吞吐率数学函数
- 使用高吞吐率数学函数
- 配置高吞吐率数学函数的输入和输出接线端

配置高吞吐率数学函数的输入和输出接线端

高吞吐率数学函数输入和输出接线端的值的范围取决于接线端的编码、字节长度和整数字节长度。如接线端有符号，值的范围是`[-2^( iwl-1), 2^(iwl-1)-2^(-wl+iwl)]`，其中：

- wl为字节长度
- iwl为整数字节长度

如接线端为无符号，值的范围为`[0, 2^ iwl -2^(-wl + iwl )]`。

例如，如wl = 16，iwl = 1，且接线端有符号，值的范围为`[-2^(1-1), 2^(1-1)-2^(-16+1)]`或`[-1, 0.999969482421875]`。如接线端为无符号，值的范围为`[0, 1.999969482421875]`。

输入接线端的不支持定点配置

接线端的定点配置是指接线端的编码、字节长度和整数字节长度。部分函数的输入接线端（例如，“高吞吐率加”函数）支持LabVIEW支持的全部定点配置。其它函数输入接线端（例如，“高吞吐率指数”函数）限制了支持的定点配置。关于函数支持的定点配置的详细信息见指定函数的帮助主题。右键单击函数，从快捷菜单中选择**帮助**访问该主题。

注： 由于编码、字节长度和整数字节长度确定了接线端的值的范围，值的范围也是影响支持的定点配置的因素。

下文列出了输入接线端的可能限制。

- 不支持的编码－例如，“高吞吐率平方根”函数不支持有符号数据类型。
- 不支持的字节长度和/或整数字节长度－例如，连线数据至“高吞吐率双曲正弦和余弦”函数，且编码的数据类型是有符号的。函数不支持大于1位的整数字节长度。
- 不支持的值的范围－例如，连线数据至“高吞吐率自然对数”函数，函数不支持超出[1/e, 1)范围的值。

在某些情况下，如果连线不支持的配置至接线端，LabVIEW将断开连线以示警告。此时不能运行函数。在某些情况下，LabVIEW强制转换连线为支持的配置。此时可运行函数。但此强制转换可能导致函数未按照预期的运行。如连线未断开或不带有强制转换点，则表示输入接线端支持该定点配置。

避免不支持的定点配置

通过使用配置对话框指定指定输入接线端的定点配置，可避免输入接线端连线断开或强制转换。按照下列步骤配置接线端。

1. 添加函数至程序框图。
2. 双击函数。LabVIEW将打开配置对话框。
3. 使用**定点配置**配置接线端的数据类型。根据使用的函数，LabVIEW禁用某些选项并限制用户可在其他文本框中的输入的值。函数支持通过配置对话框指定的特定定点配置。
4. 单击**确定**按钮，保存当前配置并关闭对话框。

完成上述步骤后，可右键单击输入接线端并选择**创建**»**输入控件**或**创建**»**常量**。创建的输入控件或常量具有函数支持的定点配置。

数组支持

某些高吞吐率数学和基本元素函数支持数组。下表汇总了支持数组的高吞吐率数学和基本元素函数。

注： 不支持数组的高吞吐率数学和基本元素函数未被列出。

函数	整数数组	定点数数组	簇数组
高吞吐率加	⊘	✓	⊘
高吞吐率减	⊘	✓	⊘
高吞吐率乘	⊘	✓	⊘
离散延迟	✓	✓	✓
累加器	⊘	⊘	⊘
加减	⊘	⊘	⊘

下列考虑因素适用于高吞吐率加、高吞吐率减和高吞吐率乘函数。

- 如两个数据输入均为数组，则数组的元素与元素执行运算。
- 如一个数据输入为数组，另一个数据输入为标量。则数组中的每个元素均与标量执行运算。
- 如数据输入为不同大小的数组，较长数组的接线端将自动调整为较短的数组大小。较长数组的接线端将出现一个强制转换点。例如，x为5个元素的数组，y为3个元素的数组。x数组中的第4个和第5个元素将被丢弃。输出为具有相同大

小的数组。

输出接线端的溢出和四舍五入

理论计算值为不考虑输出数据接线端有效范围的算术值。如理论计算值超出输出接线端的范围，将产生溢出。此时函数按照用户在配置对话框的**溢出模式**下拉菜单中指定的选项操作。用户也可指定函数是否在程序框图上显示布尔**运算溢出**输出接线端。可使用此接线端的值控制发生溢出时VI的动作。

注： 如“高吞吐率除”或“高吞吐率倒数”函数尝试除以零也会产生溢出。

多数情况下，在函数的配置对话框中勾选**匹配至源**复选框可避免溢出。如勾选此复选框，LabVIEW将尝试调整输出数据类型的宽度和范围，从而不会产生溢出或四舍五入。但此调整并不适用于所有情况。例如，函数包含除法时会一直产生四舍五入。且某些函数不包含此复选框。

相关概念：

- 使用高吞吐率数学函数

在单周期定时循环内放置高吞吐率数学函数

可在单周期定时循环内放置全部高吞吐率数学函数。但请注意下列说明：

- 多数高吞吐率数学函数是多周期的，即这些函数返回有效值的时间大于一个周期。如在单周期定时循环内放置多周期的高吞吐率数学函数，函数可能在每个时钟周期返回无效的值。函数吞吐速率和握手接线端的值确定了函数生成有效值的时钟周期。
- 如在单周期定时循环内放置几个高吞吐率数学函数，可能导致组合路径过长，编译FPGA VI时可能产生定时冲突。因此，高吞吐率数学函数提供了几种降低组合路径长度的方法。

单周期 vs. 多周期函数

高吞吐率加、减和转换为定点数函数为单周期函数。这些函数在一个FPGA时钟周期内执行。所有其他高吞吐率数学函数均为多周期函数。这些函数需要多于一个时钟周期的执行时间。如在单周期定时循环内放置几个多周期函数，在一个时钟周期内执行的组合路径可能过长。

实现高吞吐率

对于在单周期定时循环内部的多周期高吞吐率数学函数，可使用函数配置对话框内的**吞吐率**输入控件指定函数要实现的吞吐速率。以周期/采样为单位的吞吐速率为函数可接收到有效输入数据的最小FPGA时钟周期。因此**吞吐率**控件的值越小，函数等待执行的时间就越短。因为函数准备好接收有效输入数据前必须经过该数量的时钟周期。

注： 单周期函数的**吞吐率**的值总是1 周期/采样。

如要演示**吞吐率**输入控件的重要性，可考虑3个连接至握手接线端的多周期高吞吐率数学函数。3个连接的函数中的最慢的吞吐率（具有最大值的**吞吐率**输入控件）为所有连接函数可实现的最高的吞吐率。此限制对单周期定时循环内部的每个独立连接的函数均为真。

注： 仅当将函数放在单周期定时循环内部时，函数的**吞吐率**控件才可用。如在单周期定时循环外放置函数，LabVIEW用灰色显示**吞吐率**输入控件，并显示常量值为>1 调用/采样。该值表示函数在每次VI调用函数时可接收到有效的输入数据。

握手

尽管多周期高吞吐率数学函数在单周期定时循环的每个时钟周期执行，这些函数并不会在每个时钟周期返回有效的值。可配置上述函数在函数图标上显示4个握手接线端。当产生下列动作时使用这些接线端进行判定：

- 函数放弃来自上面数据流函数的数据。
- 函数接受来自上面数据流函数的数据。
- 下面数据流函数放弃来自函数的数据。
- 下面数据流函数接受来自函数的数据。

注： 判定是否显示握手接线端时，不同的函数具有不同的准则。例如，选择**在单周期定时循环内**时，高吞吐率除显示该接线端按钮。如要启用高吞吐率加函数的接线端，必须勾选**寄存器输出**复选框。

减少组合路径的长度

如在单周期定时循环内放置几个高吞吐率数学函数，组合路径的长度可能会影响FPGA VI在所需的时钟周期内完成编译。在这种情况下，编译FPGA VI时编译状态窗口将返回错误。增加流水线层级或添加输入和/或输出寄存器至函数可避免上述错误。

增加流水线层级

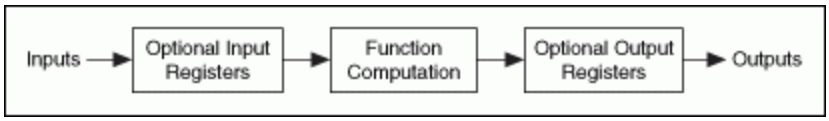
将高吞吐率复数乘函数或高吞吐率乘函数置于单周期定时循环内部，并指定较大的**流水线级数量**可缩短组合路径。指定较大的**流水线级数量**时，LabVIEW在必需的时钟速率编译函数的可能性将被增加。

添加输入和输出寄存器

也可通过为函数的输入和/或输出添加寄存器降低组合路径的长度。添加寄存器可避免编译错误。但每个寄存器组可能为函数增加一个周期的延时，也就是说接收有效输出值的时间增加了一个周期。

注： 部分函数（例如，“高吞吐率减”函数）支持仅添加一组寄存器。

下图为数据流经包含上述内部寄存器的函数的示意图。



使用配置对话框的**寄存器**部分添加输入和/或输出寄存器。

注： 如在单周期定时循环外部放置函数，LabVIEW编译VI时，将自动在函数输出端放置寄存器。

相关概念：

- 缩短FPGA VI的组合路径
- 使用高吞吐率数学函数

使用线性代数函数

使用“线性代数”函数在高速和高吞吐率的FPGA应用（例如，RF应用）中执行向量和矩阵运算。用户可配置“线性代数”函数匹配不同的吞吐率、矩阵/向量大小和数据类型，以满足定时和资源要求。

线性代数动作

使用“线性代数”函数开发FPGA应用时，请考虑下列动作：

- **终端支持** – 全部FPGA终端均支持线性代数函数。
- **单周期定时循环支持** – LabVIEW仅在单周期定时循环内部支持线性代数函数。
- **4线握手协议支持** – 线性代数函数支持握手信号。
- **仿真支持** – 线性代数函数支持桌面和全框图仿真。但当**输出有效**为FALSE时，不能确保桌面仿真的周期精确行为，因为函数未计算出下游节点可使用的结果。**输出有效**为FALSE时，函数在FPGA终端上可能返回与主机不同的结果。**输出有效**终止时返回的数据可能无效且应该丢弃。
- **受限的数据类型支持** – 线性代数函数仅支持定点数据类型的标量和向量值。每个函数包含用于实部和虚部的独立连线板接线端。
- **带标签的输入和输出接线端** – 可在**即时帮助**窗口中查看每个函数数值接线端的

编码、字长和整数字长。

配置“线性代数”函数

如要配置线性代数函数，可双击函数，或右键单击函数并选择**配置**。LabVIEW将显示函数的配置对话框。使用该对话框配置函数执行的方式和函数返回的结果。如要避免不支持的定点配置（例如，不支持的字节长度或整数字节长度）导致连线断开，使用配置对话框设置输入接线端的编码、字节长度和整数字节长度。

配置函数外观

每个“线性代数”函数可使用展开或折叠的方式显示。展开视图可显示每个接线端的编码、字节长度和整数字节长度。折叠视图可节省程序框图的空间。如要配置函数的外观，右键单击函数并选择**展开视图**或**折叠视图**。

输入和输出接线端的说明和建议

下表介绍了“线性代数”函数的输入和输出接线端的动作：

- “线性代数”函数仅支持定点数据类型的标量和向量输入。每个函数包含用于实部和虚部的独立接线端。
- “线性代数点积”、“线性代数矩阵转置”和“线性代数范数平方”函数的输入接线端匹配数据类型及连线至函数的输入模式。“线性代数矩阵乘法”的输入接线端匹配至数据类型，但不会匹配连线至函数的输入模式。使用**配置线性矩阵乘法**对话框配置线性代数矩阵乘法函数的输入模式。
- “线性代数”函数的实部和虚部输入必须具有相同的数据类型和输入模式，否则将使用输入接线端的实部数据类型和数组大小。

输入接线端不支持的配置

如连线不支持的定点配置至接线端，LabVIEW则显示为连线断开且无法运行函数。LabVIEW可能强制转换数据类型为支持的配置。此时可运行函数。但此强制转换可能导致函数未按照预期的运行。如连线未断开且不带有强制转换点，则表示输入接线端支持该定点配置。

避免不支持的输入配置

通过使用配置对话框指定特定输入接线端的定点配置，可避免输入接线端连线断开或强制的数据类型转换。按照下列步骤配置接线端：

1. 添加函数至程序框图。
2. 双击函数打开配置对话框。
3. 使用配置对话框的输入区域，配置输入接线端的编码、字节长度和整数字节长度。根据使用的函数，LabVIEW禁用某些选项并限制用户可在其他文本框中的输入的值。
4. 单击**OK**按钮。

右键单击输入接线端，选择**创建»输入控件**或**创建»常量**创建采用函数支持的定点配置的输入。

输出接线端的溢出和四舍五入

理论计算值为不考虑输出数据接线端有效范围的算术值。对于包含**溢出模式**配置的线性代数函数，如理论计算值超出输出接线端的数据范围，则产生溢出。此时函数按照用户在配置对话框的**溢出模式**下拉列表中指定的选项操作（绕回或饱和）。

多数情况下，在函数的配置对话框中勾选**匹配至源**复选框可避免溢出。如启用此选项，LabVIEW将尝试调整输出数据类型的宽度和范围，从而不会产生溢出或四舍五入。但此调整并不适用于所有情况。LabVIEW支持最大64位的字长和最大1023位的整数字节长度。如勾选该复选框且输出数据类型需要超出最大值范围的字长，可能产生溢出和/或凑整错误。

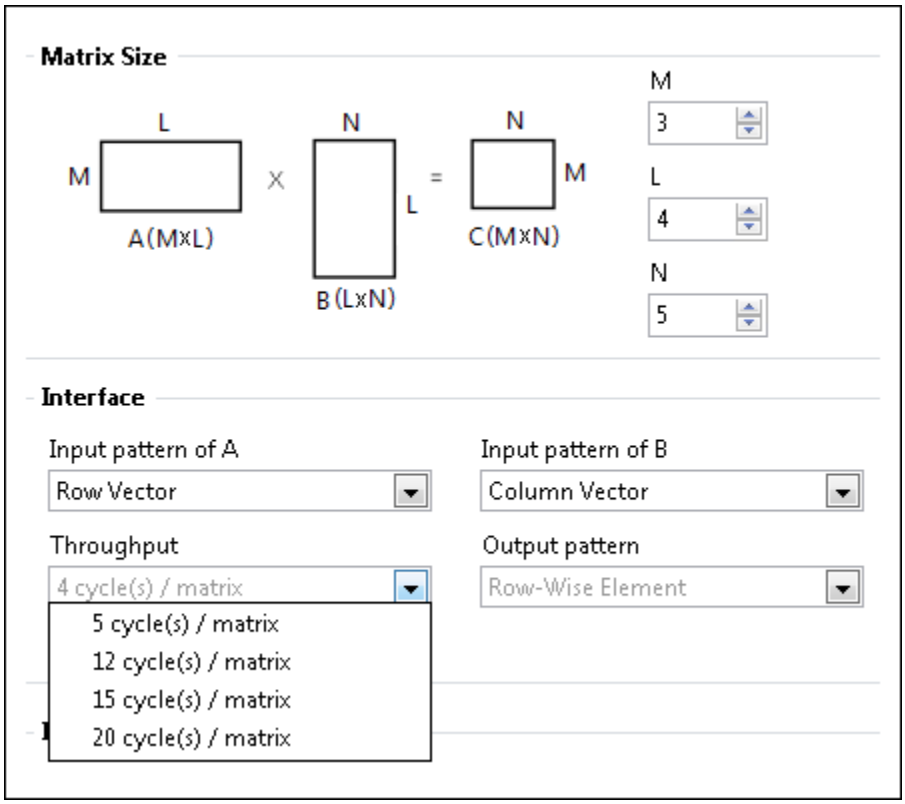
配置线性代数矩阵乘法函数的矩阵大小和接口

为使“矩阵乘法”函数对输入数据执行操作，必须使用**配置线性代数矩阵乘法**对话框配置函数。按照下列步骤配置“线性代数矩阵乘法”函数的矩阵大小和接口：

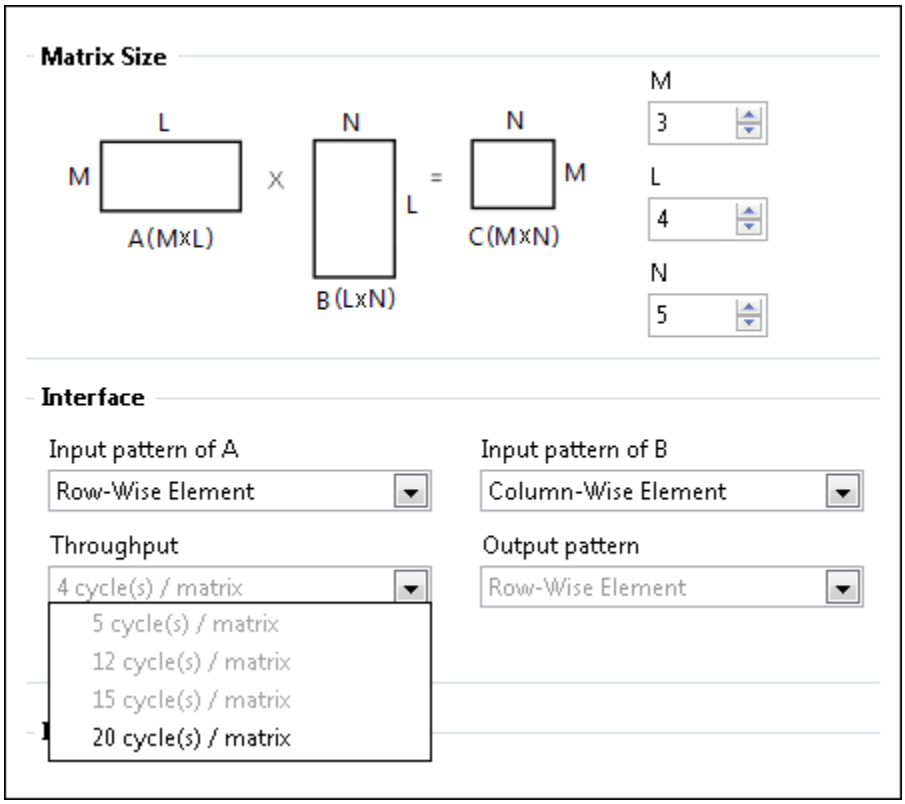
1. 在**矩阵大小**部分输入**M**、**L**和**N**的值，以设置矩阵A和矩阵B的大小。
2. 在**接口**配置部分选择**A输入模式**和**B输入模式**，指定函数接收矩阵元素的方式：

向量、行为主或列为主。矩阵A和矩阵B必须同时使用相同的输入模式。例如，如果**A输入模式**为**行为主元素**或**列为主元素**，则可以只选择**行为主元素**或**列为主元素**作为**B输入模式**，LabVIEW将以灰色显示向量输入模式。同样，如果**A输入模式**为**行向量**或**列向量**，则可以只选择**行向量**或**列向量**作为**B输入模式**，LabVIEW将以灰色显示元素输入模式。

3. 使用**吞吐量**选项指定需要函数实现的吞吐速率。吞吐率（周期/矩阵为单位）为函数可接收下一对有效输入矩阵前必须经过的最小FPGA时钟周期。函数需要几个时钟周期计算矩阵A与矩阵B的乘法结果。输入下一对矩阵至下一个函数前，需要等待几个时钟周期。**吞吐量**选项根据**M**、**L**和**N**配置的矩阵大小变化。例如，如果矩阵A的大小为3*4 (**M***L)并采用**行向量**输入模式，矩阵B的大小为4*5 (**L***N)并采用**列向量**输入模式，则**吞吐量**可能为3、4、5、12、15或20周期/矩阵。但由于矩阵A的输入周期为3 (**M**)，矩阵B的输入周期为5 (**N**)，可用的**吞吐量**应大于或等于5周期/矩阵。因此，3 周期/矩阵和4 周期/矩阵无效。LabVIEW显示可用的**吞吐量**选项：5、12、15和20周期/矩阵。通常选择较小值的周期/矩阵可产生较大的吞吐率，但同时消耗较多的FPGA资源。下列对话框为用于范例的正确配置：



在下列范例中，如矩阵A被配置为**行为主元素**，矩阵B被配置为**列为主元素**，LabVIEW则显示可用的**吞吐量**选项为：3、4、5、12、15或20周期/矩阵。矩阵A的输入周期为12 (**M*L**)，矩阵B的输入周期为20 (**L*N**)。可用的**吞吐量**应大于或等于矩阵A和矩阵B的最大输入周期。本范例中为20周期/矩阵。因此，可用的**吞吐量**选项为20周期/矩阵。下列对话框为用于范例的正确配置：

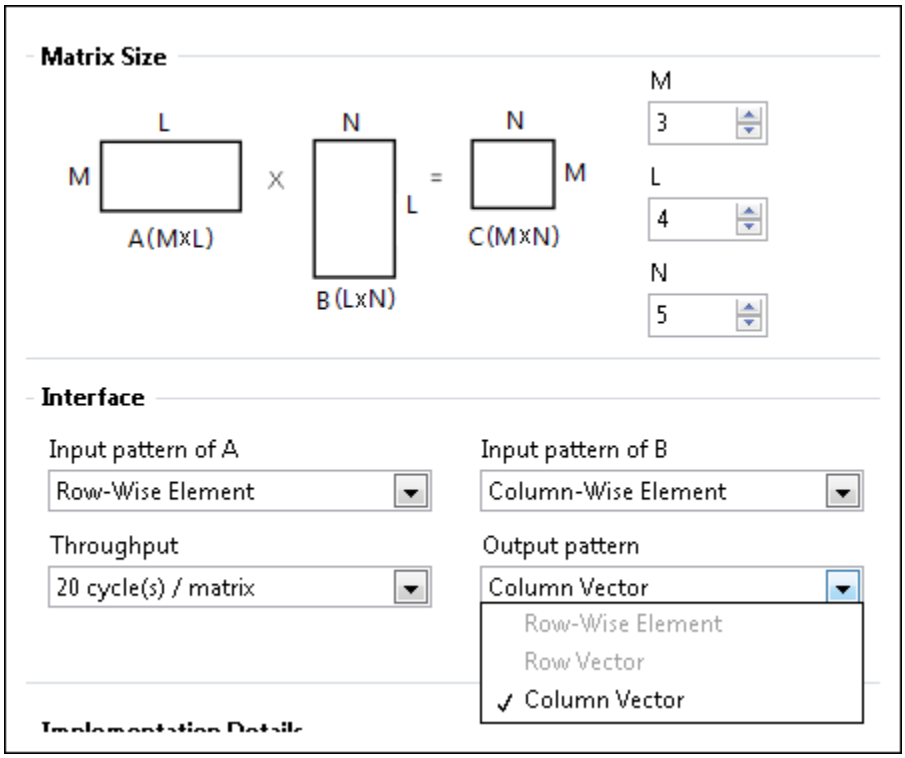


4. 选择函数的**输出模式**。线性代数矩阵乘法函数的可能吞吐量为**M**、**L**、**N**、**M*L**、**N*L**和**M*N**。每个吞吐量均对应特定的输出模式。关于吞吐量和**输出模式**的对应关系见下表。

吞吐量（以周期/矩阵为单位）	**输出模式**
M	**行向量**
L	**列向量**
N	**列向量**
M × L	**行向量**
L × N	**列向量**

吞吐量（以周期/矩阵为单位）	输出模式
M × N	行为主元素

在以下范例中，矩阵A的大小为3*4 (**M*L**)，矩阵B的大小为4*5 (**L*N**)，因此**吞吐量**为20周期/矩阵(**L*N**)。因此，A*B矩阵的**输出模式**为**列向量**。下列对话框为用于范例的正确配置：



使用整型数据类型

通过数值函数在FPGA上执行整数运算。

执行整数数学运算时，结果可能溢出。数学运算结果超出输出数据类型时产生溢出。例如，U8整型的范围是0~255。同时添加2个U8整型产生的结果将超过255（例如，200 + 70）。产生溢出时，当达到数据的最大限制时，结果将被回零重新计数或回绕并返回256的模值。例如，U8整数的结果为270时，将在256时发生回绕并返回结果14。

某些应用中可以利用溢出的回零特性。例如，执行时间测量范例就依赖于溢出的回

零特性来正常工作。范例为时间计数器 Express VI配置一个8位的**内部计数器大小**及以毫秒为单位的**计数器单位**。时间计数器Express VI的内部计数器达到255 ms时，计数器回零。如第一个时间计数器Express VI返回**时间计数器**的值为132 ms，待测量的LabVIEW代码的执行时间为140 ms。内部计数器已经回零，且第二个时间计数器Express VI返回**时间计数器**的值为16 ms。程序框图从132中减去16，此时将产生溢出且结果值为140。

注： 时间计数器Express VI的执行时间为一个周期。在该范例中，如设置**计数器单位**为**Ticks**而不是**mSec**，则减法返回的值为141。即使中间序列的LabVIEW代码仅需要140个时钟滴答时间。

使用下列技术避免整数溢出：

- 评估数值数据类型的使用量。
- 使用较大的输出数据类型。使用较大的输出数据类型时会占用较多的FPGA终端空间，但能够更快和更便捷的编程FPGA VI，且收到的数据更加精确。
- 使用“按2的幂缩放”函数减少定点数据输入的整数幅值。如使用带有常量**n**输入的“按2的幂缩放”函数，该换算不占用任何FPGA资源。但损失了精度并且用户需要仔细编程FPGA VI，以确保所有输入和输出端的值均进行了正确的缩放。
- 使用“转换为单精度浮点数”函数转换整型数据为单精度浮点型。

相关概念：

- 使用FPGA定时函数管理执行速率

使用单精度浮点型数据类型

单精度浮点型 (SGL)数据相比24位定点数据类型能够提供更高的精度，但由于其增加了函数延时且占用较多的FPGA资源，系统整体性能将被降低。评估数值数据类型的使用量，以确定最适合用户设计的数据类型。

下表详细介绍了使用单精度浮点型数据开发FPGA应用时的应注意事项：

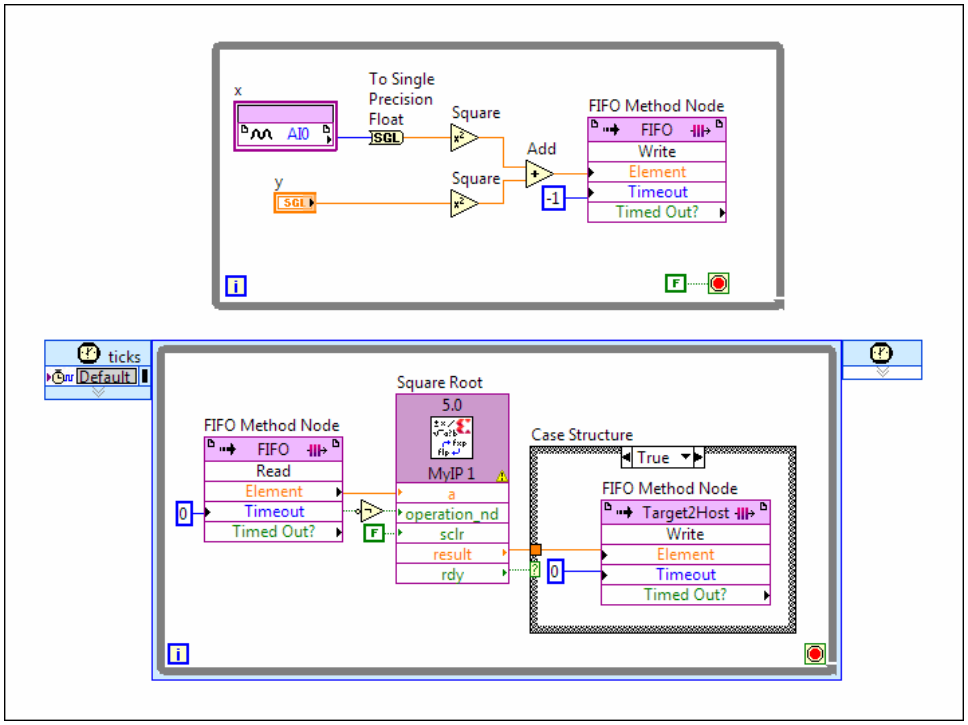
- FPGA模块不支持双精度或扩展精度浮点数。
- 全部FPGA终端均支持SGL数据类型。
- 仅特定函数支持SGL数据类型。
- 高吞吐率数学函数不支持SGL数据类型。
- 由于多数函数均需要大于1个时钟周期的执行时间，且不具有握手信号，因为它们仅能在单周期定时循环外执行单精度浮点型运算。
- FPGA VI中的数据类型强制转换将占用大量的FPGA资源，尤其是接线端被强制转换为SGL数据类型。
- FPGA模块不支持在单周期定时循环内强制转换为SGL数据类型。
- 在FPGA模块中，SGL数据类型与IEEE Std 754-2008兼容，但其不支持非规格化数值。例如，如使用“等于？”函数比较主机和终端的位精度，如运算为非常小的数值，函数将返回FALSE结果。上述错误也可能在反馈循环内传输并降低数据精度。
- FPGA的NaN输出的位模式可能与开发计算机的NaN输出不同。
- FPGA上执行的单精度浮点型运算结果可能与在仿真中执行的单精度浮点型运算结果不同。

单周期定时循环内的单精度浮点型运算

由于多数函数均需要大于1个时钟周期的执行时间，且不具有握手信号，因为它们仅能在单周期定时循环外执行单精度浮点型运算。下列功能为在单周期定时循环内部使用单精度浮点型数据支持的功能：

- IP集成节点
- Xilinx IP
- 存储器项
- FIFO
- 寄存器
- 局部变量和全局变量
- 非法数字/路径/引用句柄？函数
- 选择函数

关于使用FIFO传递单精度浮点型数据至单周期定时循环的范例，见下列FPGA VI。



顶层循环显示了数据流经单周期定时循环外部的“平方”和“加”函数。即时结果随后通过FIFO方法节点被传输至单周期定时循环，以计算数据的平方根。最后DMA FIFO传输结果至主计算机。

设计带有单精度浮点型数据类型的FPGA应用

设计使用单精度浮点型运算的FPGA应用时，请考虑下列因素。

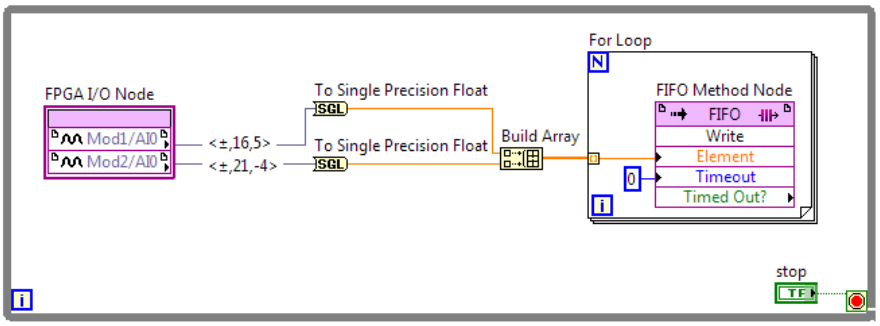
释放主机处理器以执行其他运算，并使用单精度浮点型与I/O转换

执行I/O至FPGA的转换以释放主机空间，尤其针对实时系统。如具有可用的FPGA资源，在FPGA上执行定点I/O至单精度浮点型数据转换释放主机处理器，以执行其他运算。例如，满足实时要求。

执行多个输入或输出端的运算

转换不同的数据路径为SGL数据类型，以使用相同的代码处理数据。对于需要处理来自多个输入端，且具有不同定点数据类型的运算，该转换非常有用。

下列FPGA VI执行下列操作：NI cRIO-9104从两个模块获取数据并将其转换为SGL数据类型，数据被写入至DMA FIFO。



DSP48E和DSP48E1逻辑片简介

注： 该帮助文档并非DSP48E或DSP48E1逻辑片的详尽说明。使用这些函数前，建议先熟悉***Virtex-5 FPGA XtremeDSP设计考虑要素用户指南***中DSP48E函数的相关信息及***Virtex-6 FPGA DSP48E1逻辑片用户指南***中DSP48E1函数的相关信息。上述指南可通过Xilinx网站www.xilinx.com获取，且其包含了逻辑片的详细信息。涉及内容有：架构的详细信息、定时考虑因素、输入和输出端口的说明及帮助用户有效编程逻辑片的范例应用。

DSP48E逻辑片是特定FPGA系列（例如，Xilinx Virtex-5）包含的数字信号处理逻辑单元。该逻辑片可用于实现不同的算术运算。例如，乘法累加器、乘加器、单步/n步计数器。该逻辑片还可用于执行AND、OR、XOR等逻辑运算。使用多个DSP48E逻辑片可在不占用额外FPGA架构资源的前提下实现更为复杂的功能。例如，实现复杂乘法器或n阶FIR滤波器。

DSP48E1是某些Xilinx Virtex-6或更高级别FPGA设备的数字信号处理单元。DSP48E1在多个方面扩展了DSP48E逻辑片的功能（例如，在乘法器前包含一个预加器）。该预加器对执行特定的算法非常有用（例如，对称FIR滤波器）。Virtex-6 FPGA DSP48E1 Slice User Guide的第11页中详细介绍了增强的功能。

如要访问DSP48E逻辑片，请先添加DSP48E函数至程序框图，然后配置函数。

相关概念：

- 配置DSP48E和DSP48E1函数

配置DSP48E和DSP48E1函数

配置DSP48E函数或DSP48E1函数包含下列步骤：

1. 配置逻辑片的功能，包括乘法器、预加器（仅限DSP48E1）、加/减法器或逻辑单元和模式检测器
2. 通过添加或移除内部寄存器调整内部组合路径的长度
3. 判定要显示的程序框图接线端
4. 确保这些接线端的精确度
5. 查看内部数据路径的延迟

下列章节提供了补充信息：

- 使用上述函数的说明
- 缩小函数图标
- 确定DSP48E或DSP48E1函数的最大可用数量
- 使用DSP48E1函数替代DSP48E函数或反之

教程

参阅***DSP48E范例：创建复数乘法器***和***DSP48E范例：创建一个n阶FIR滤波器***教程，以熟悉这些函数。

相关概念：

- 配置DSP48E或DSP48E1逻辑片的功能
- 配置DSP48E或DSP48E1函数的模式检测
- 调整DSP48E或DSP48E1函数的内部组合路径的长度
- FPGA硬件概述

- 显示和隐藏DSP48E或DSP48E1函数的接线端
- 查看DSP48E和DSP48E1函数的数据路径延迟
- 使用DSP48E和DSP48E1函数的说明
- DSP48E范例：创建复数乘法器
- DSP48E范例：创建一个n阶FIR滤波器
- DSP48E和DSP48E1逻辑片简介

配置DSP48E或DSP48E1逻辑片的功能

添加DSP48E或DSP48E1函数至程序框图后，双击函数打开其配置对话框。第一个页面为**函数**页面，用于配置函数的总体信息。

注： 注意配置对话框底部的示意图。此示意图将出现在每个页面上，并随用户配置函数更新。以提供函数逻辑的可视化表达方式。

opmode、**alumode**和**carryinsel**输入端的值决定了DSP48E和DSP48E1逻辑片的功能。**inmode**输入端的值将影响DSP48E1逻辑片。在**函数**页面的下拉菜单中选择**算术配置**或**逻辑配置**，通过LabVIEW配置接线端。此时用户按照下列章节配置算术或逻辑表达式后，LabVIEW将设置相应的值。

注： 如要查看LabVIEW为这些输入端指定的值，单击**VHDL实例**选项卡。

配置用于算术表达式的DSP48E或DSP48E1函数

如选择**算术配置**，函数计算下列算术表达式：

p = (+/NOT **z**) +/– (**x+y** + **carryin**)

其中，

- 非**z**等于–**z** – 1。
- **z**、**x+y**和**carryin**为变量。用户为其选定为0或数据源（例如，函数接线端或运算的值）的常量值。用户为参数指定的值和/或源将决定逻辑片的功能。通过将

z或x+y的值指定为0，可从等式中移除z或x+y。

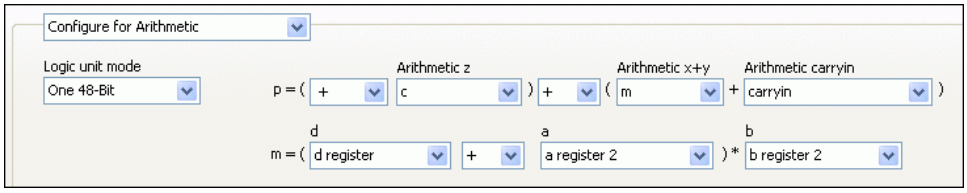
注：

- carryin可用源取决于用户为z和x+y指定的值。因此建议在指定carryin前，指定z和x+y的值。
- (DSP48E)关于carryin的不同的源的相关信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***中的表1-10。通过www.xilinx.com可获取该文档。手册中将此接线端标记为CARRYINSEL。(DSP48E1)关于该函数的信息，见***Virtex-6 FPGA DSP48E1 Slice User Guide***中的表1-11。

- p为等式的数值结果。

配置DSP48E1逻辑片的预加器

如要使用DSP48E1逻辑片的预加器，设置x+y的值为m，其中m = (d + a) × b。d、a和b可为零或d、a和b输入接线端路径中的不同阶。下列对话框为使用预加器的配置范例：



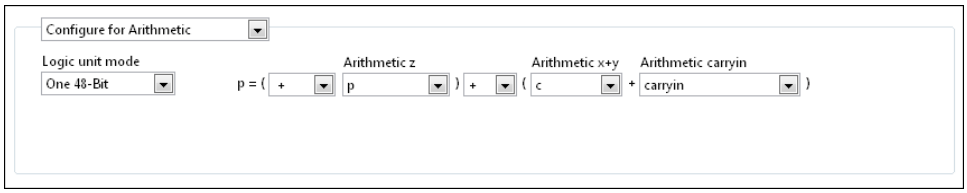
关于预加器的详细信息，见***Virtex-6 FPGA DSP48E1 Slice User Guide***的下列章节。

- "预加器"
- "预加器块应用"

DSP48E的常用算术配置

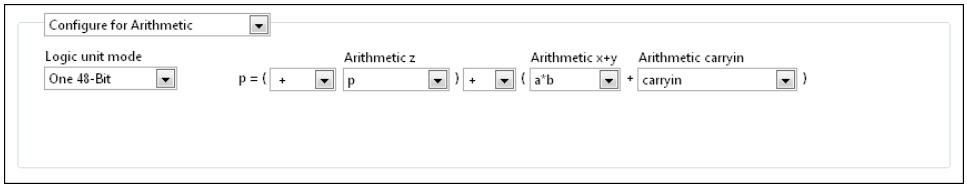
下列对话框为DSP48E函数的常用算术配置：

乘

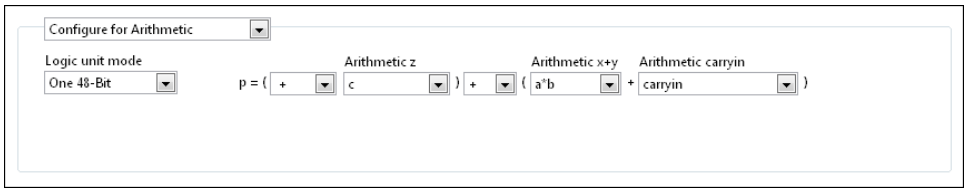


注： 对于所有包含乘法器的配置，必需设置**逻辑单元模式**为**1个48位**。

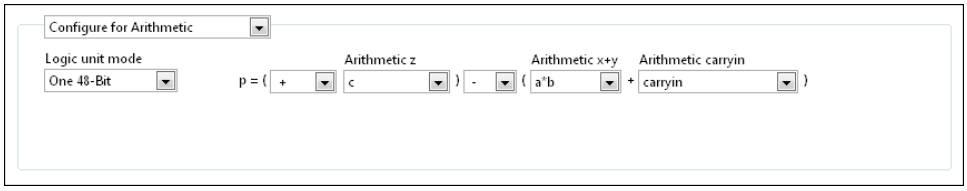
乘累加器



乘加器

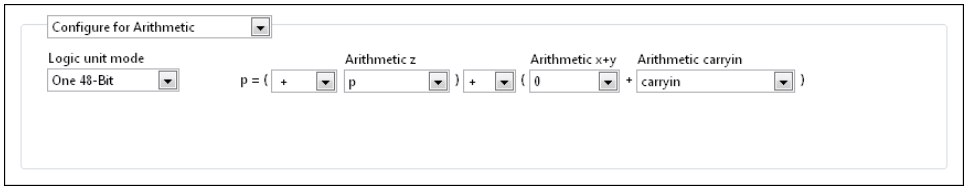


乘减法器

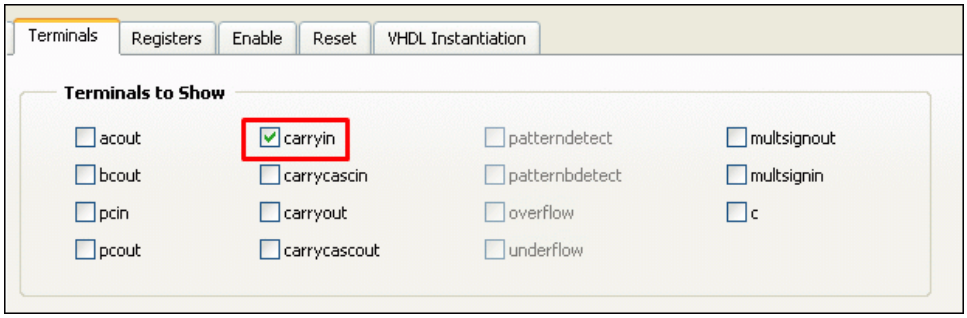


一阶计数器

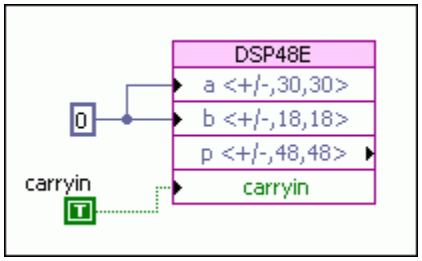
1. **函数**页面：



2. **接线端**页面：

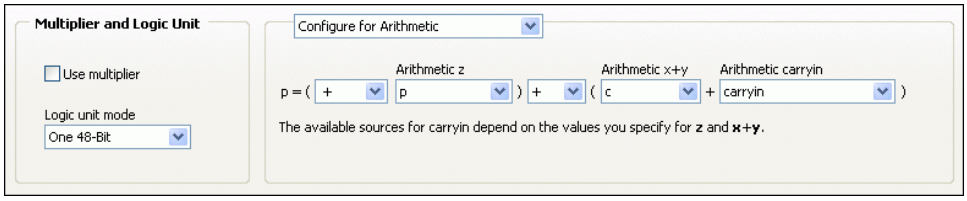


3. 在程序框图上，连线TRUE常量至**carryin**接线端，0至**a**和**b**输入接线端：

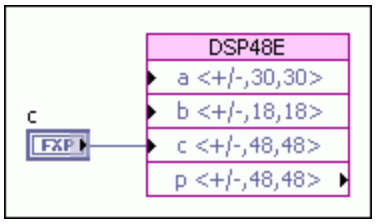


n阶计数器/累加器

1. **函数**页面：



2. 在程序框图上，连线控件至**c**输入端：



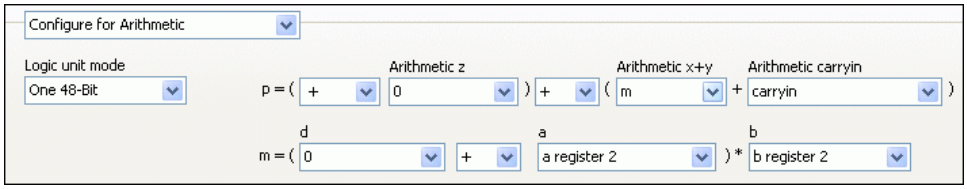
c代表计数器的阶数。函数累加c的值。

注： 在此配置中未显示carryin接线端，表示LabVIEW将其常量值设置为FALSE。

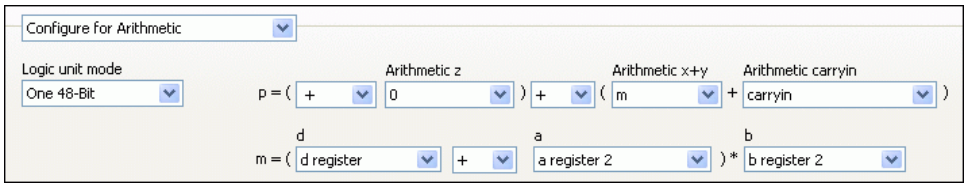
DSP48E1的常用算术配置

下列对话框为DSP48E1函数的常用算术配置：

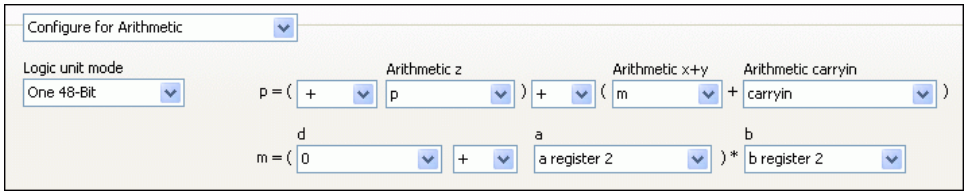
乘



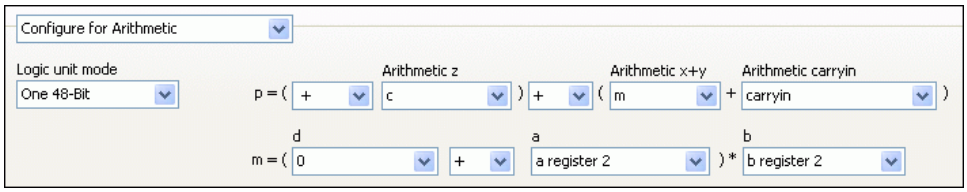
带预加器的乘



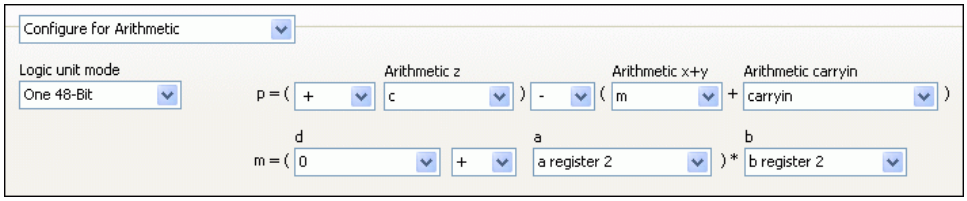
乘累加器



乘加器

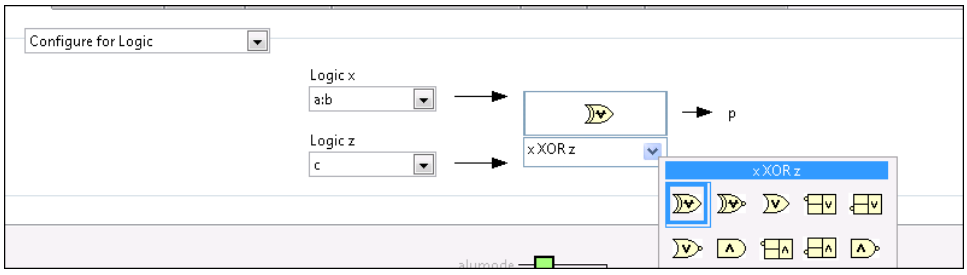


乘减法器



配置用于逻辑表达式的DSP48E或DSP48E1函数

选择**逻辑配置**时，函数比较**逻辑x**和**逻辑z**，并返回结果**p**。必须指定**逻辑x**、**逻辑z**以及比较运算，如下图所示：



VI运行时，配置DSP48E或DSP48E1函数的动作

如要在VI运行时更改DSP48E或DSP48E1逻辑片的动作，首先必须在配置对话框中选择**自定义配置**。选择此选项后，LabVIEW在程序框图上显示**opmode**、**alumode**和**carryinsel**接线端。LabVIEW也会显示用于DSP48E1函数的**inmode**接线端。必须连线值到这些接线端以实现所需的配置。由于该配置是通过编程实现的，可在FPGA VI运行时改变DSP48E或DSP48E1逻辑片的动作。

注： 如连线无效的值至这些接线端，FPGA VI的动作可能超出预期。

(DSP48E)关于有效值列表以及值对DSP48E逻辑片的影响，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***的下列章节。通过Xilinx网站www.xilinx.com可获取该文档。

- “ALUMODE输入”
- "进位输入逻辑"
- “两输入逻辑单元”
- “X、Y和Z多路复用器”

(DSP48E1)详细信息，见***Virtex-6 FPGA DSP48E1 Slice User Guide***的"A、B、C和D端口"章节。

等效于LabVIEW选项的VHDL代码

如要查看配置的相应VHDL代码，单击**VHDL实例**选项卡。关于指定属性的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***或***Virtex-6 FPGA DSP48E1 Slice User Guide***中的表1-3。通过Xilinx网站www.xilinx.com可获取该文档。下表为对应于LabVIEW中每个选项的属性。

注： 关于该属性的详细信息，见下列章节：

- "Single Instruction, Multiple Data (SIMD) Mode"
- "Single Instruction Multiple Data (SIMD) Arithmetic"

相关概念：

- 配置DSP48E和DSP48E1函数
- 配置DSP48E或DSP48E1函数的模式检测

配置DSP48E或DSP48E1函数的模式检测

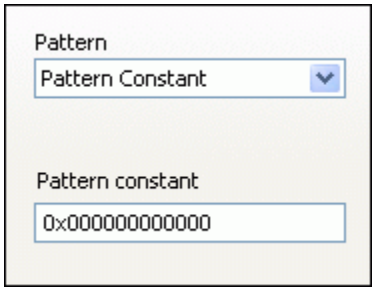
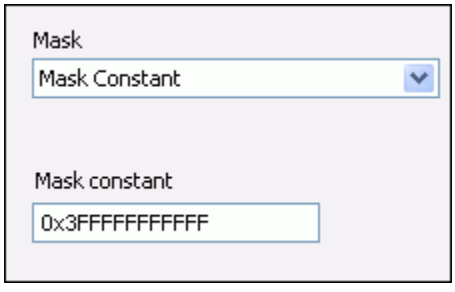
通过**模式检测**页面启用、禁用及配置模式检测。关于模式检测的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***或***Virtex-6***

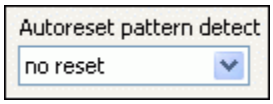
FPGA DSP48E1 Slice User Guide的下列章节。通过Xilinx网站www.xilinx.com可获取该文档。

- "Pattern Detect Logic"
- "PATTERNDETECT and PATTERNBDETECT Port Logic"
- "Pattern Detect Applications"（仅限***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***）。本节还提供了下文表中的VHDL类属代码和选项的信息。

等效于LabVIEW选项的VHDL代码

如要查看配置的相应VHDL代码，单击**VHDL实例**选项卡。关于指定属性的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***或***Virtex-6 FPGA DSP48E1 Slice User Guide***中的表1-3。通过Xilinx网站www.xilinx.com可获取该文档。下表为对应于LabVIEW中每个选项的属性。

LabVIEW选项	表1-3的参考属性	
	DSP48E	DSP48E1
	USE_PATTERN_DETECT	
	SEL_PATTERN, PATTERN	
	SEL_ROUNDING_MASK, SEL_MASK, MASK	SEL_MASK, MASK

LabVIEW选项	表1-3的参考属性	
	DSP48E	DSP48E1
	AUTORESET_PATTERN_DETECT, AUTORESET_PATTERN_DETECT_OPTINV	AUTORESET_PATDET

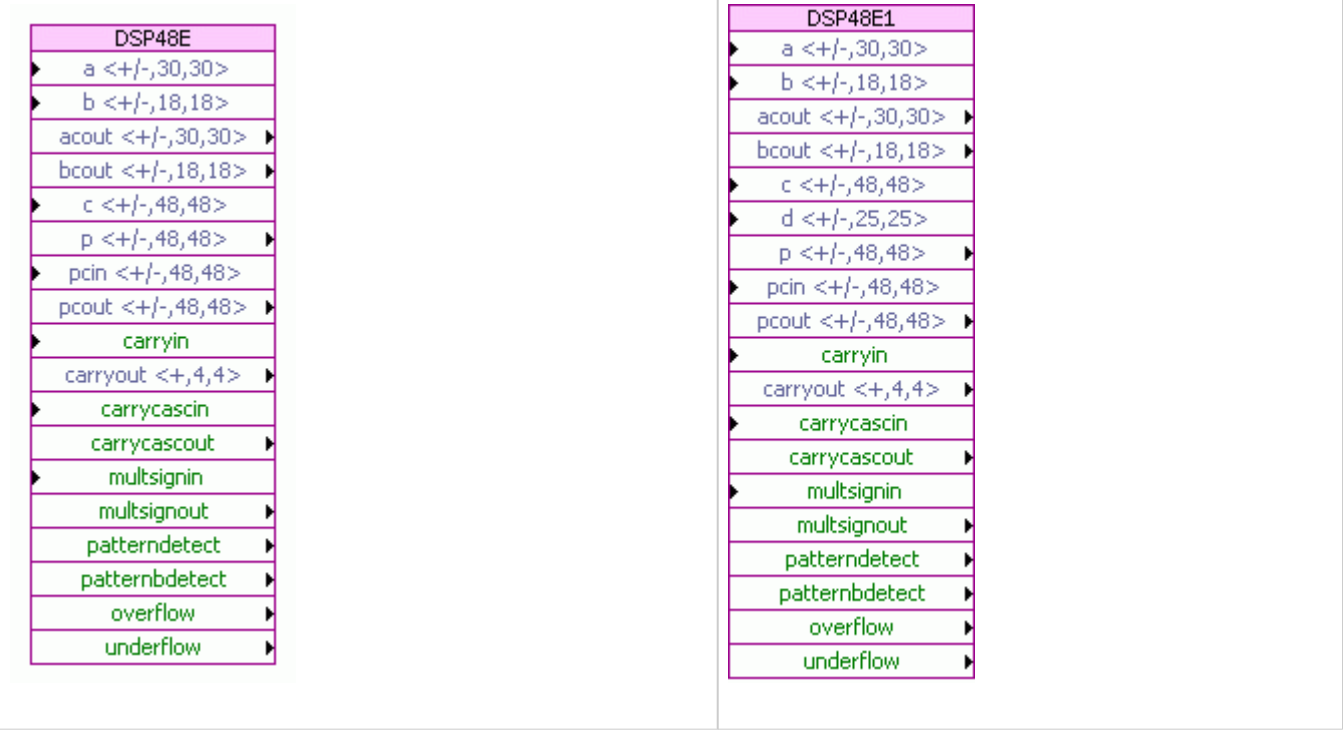
相关概念：

- 配置DSP48E和DSP48E1函数
- 配置DSP48E或DSP48E1逻辑片的功能
- 显示和隐藏DSP48E或DSP48E1函数的接线端

显示和隐藏DSP48E或DSP48E1函数的接线端

添加DSP48E或DSP48E1函数至程序框图后，LabVIEW仅显示**a**、**b**、**c**和**p**接线端。其中**p**为函数的输出端，其他为输入接线端。用户也可以选择显示其它接线端。

通过配置对话框的**接线端**页面指定LabVIEW显示的程序框图接线端。下表为2个函数的全部接线端名称和数据类型。



所有数值型接线端均为定点型。上图显示了接线端的编码、字节长度以及整数字节长度。如隐藏了这些接线端，LabVIEW将设置其值为0。这些接线端不会匹配至源；如连线不同数据类型的定点输入控件或显示控件至接线端，LabVIEW将强制转换值。这将降低数据精度。如要保持精度，可连线一个具有相同数据类型的值至接线端，或缩放接线端的整数字长。

注： 连线指定输出接线端时，请注意说明。

其他接线端为布尔型。如隐藏了这些接线端，LabVIEW将设置其值为FALSE。仅在启用模式检测时，**patterndetect**和**patternbdetect**接线端可用。仅在为**p**启用寄存器时，**overflow**和**underflow**接线端可用。

如显示一个输入接线端（例如，**c**），相应的VHDL为`C => C`。如在其他地方使用该VHDL代码，必须使用驱动**c**输入端的信号名称替换参数右侧的`C`。如隐藏该接线端，LabVIEW和VHDL均设置**c**的值为常量0。

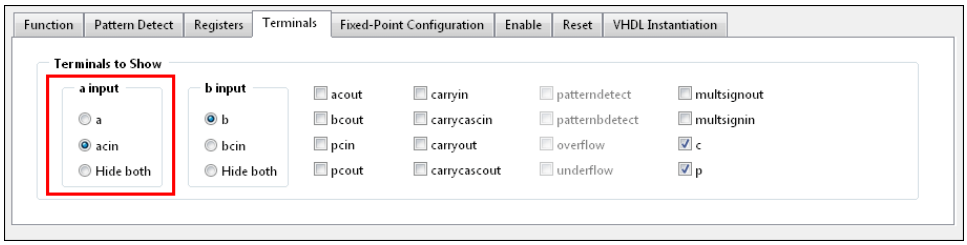
如显示一个输入接线端（例如，**overflow**），相应的VHDL为`OVERFLOW =>`

OVERFLOW。如在其他地方使用该VHDL代码，必须替换参数右侧的信号名称为OVERFLOW驱动的信号的名称。如隐藏该接线端，VHDL代码为OVERFLOW => OPEN。

注： 关于函数输入端和输出端的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***的"Input Ports"和"Output Ports"章节。通过Xilinx网站www.xilinx.com可获取该文档。(DSP48E1)关于仅DSP48E1函数可用的输入和输出的详细信息，见***Virtex-6 FPGA DSP48E1 Slice User Guide***的相应部分。

级联或隐藏a和b输入接线端

使用**接线端**页面的**要显示的接线端**部分级联或隐藏a和/或b输入接线端。下图显示了该部分：



通过该部分完成下列任务：

- 如设置接线端为级联模式，分别选择**acin**或**bcin**选项。该函数则显示**acin**或**bcin**接线端，而不会显示**a**或**b**接线端。必须连线**acin**和**bcin**至另一个DSP48E或DSP48E1函数的**acout**或**bcout**输出接线端

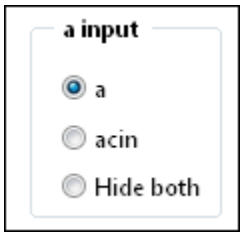
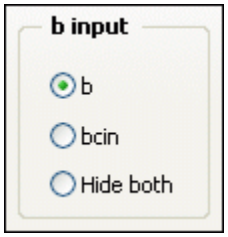
  注： **acin**与**a**输入接线端具有相同的数据类型。**bcin**与**b**输入接线端具有相同的数据类型。上述接线端的字节长度是固定的，但可调整整数字节长度以保持精度。

- 如要设置接线端为直连模式，分别选择**a**或**b**选项。
- 如不使用接线端，可选择**隐藏全部**隐藏接线端选项。

关于级联及DSP48E和DSP48E1逻辑片的意义的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***或***Virtex-6 FPGA DSP48E1 Slice User Guide***。通过Xilinx网站www.xilinx.com可获取该文档。

等效于LabVIEW选项的VHDL代码

如要查看配置的相应VHDL代码，单击**VHDL实例**选项卡。关于指定属性的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***或***Virtex-6 FPGA DSP48E1 Slice User Guide***中的表1-3。通过Xilinx网站www.xilinx.com可获取该文档。下表为对应于LabVIEW中每个选项的属性。

LabVIEW选项	表1-3的参考属性
	A_INPUT
	B_INPUT

注： 用于b和bcin接线端的VHDL类似，除类属为B_INPUT、B和BCIN。

相关概念：

- 调整DSP48E或DSP48E1函数的内部组合路径的长度
- 使用DSP48E和DSP48E1函数的说明
- 配置DSP48E和DSP48E1函数

- 配置DSP48E或DSP48E1函数的模式检测
- 保持DSP48E和DSP48E1函数接线端的精度

调整DSP48E或DSP48E1函数的内部组合路径的长度

DSP48E或DSP48E1配置对话框底部的示意图显示了函数的内部组合路径。如该组合路径过长，函数不能在用户指定的时钟周期编译。尝试编译FPGA VI时，LabVIEW将返回错误。

为函数添加内部寄存器可避免发生上述错误。添加寄存器可增加函数在指定时钟周期内完成编译的机率。但也增加了DSP48E函数的延迟，可能导致内部数据路径延迟的相对不平衡。根据使用函数的方式，该不平衡是可以接受的。但必须考虑内部数据路径的延迟，以确保函数能够按照预期执行。

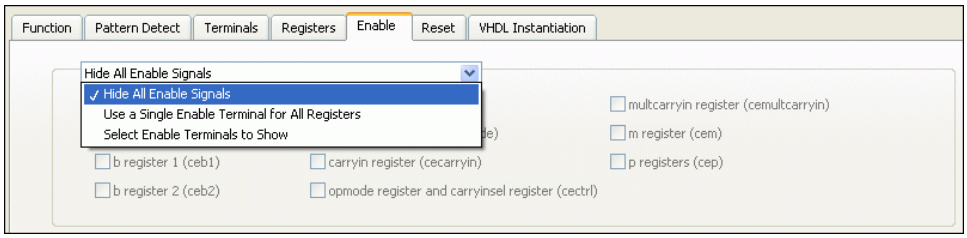
通过配置对话框的**寄存器**页面添加和配置DSP48E逻辑片的内部寄存器。注，添加和配置寄存器时，对话框底部示意图中的绿色框亮起。数据路径也可能发生改变。

> **注：** 内部寄存器仅在顶层FPGA VI运行时重置。(DSP48E)关于逻辑片内寄存器的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***中的表1-4。通过`www.xilinx.com`可获取该文档。(DSP48E1)关于该函数的信息，见***Virtex-6 FPGA DSP48E1 Slice User Guide***中的表1-4。

启用和重置寄存器

每个寄存器都具有相应的启用和重置接线端。默认情况下，LabVIEW设置重置接线端为FALSE，且从程序框图隐藏所有接线端时启用接线端为TRUE。如默认设置可用，则无需更改设置。

如要详尽的控制启用和充值操作，可使用**启用**和**重置**页面指定LabVIEW在程序框图上显示的启用和重置接线端。在每个页面的顶端为可选选项的下拉菜单，如下图所示：



这些选项对添加的寄存器的影响如下：

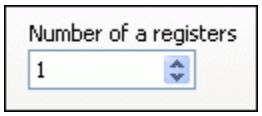
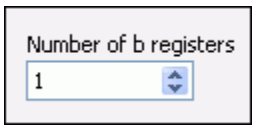
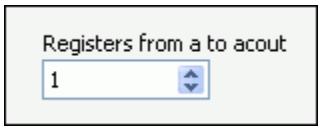
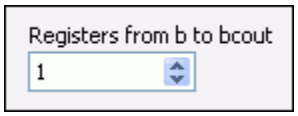
- **隐藏全部信号**－指定LabVIEW在程序框图上不显示启用或重置接线端。此时LabVIEW执行上述的默认操作。
- **所有寄存器使用一个接线端**－根据用户的配置页面，指定LabVIEW显示**启用**或**清除**输入接线端。如要在逻辑片中启用/重置所有寄存器，连线一个TRUE值到该接线端。
- **选择要显示的接线端**－启用该下拉菜单下的复选框。如要显示单个寄存器的接线端，请勾选相应的对话框。括号中的名称为启用或重置信号的VHDL名称。

> **注：** 如选择了该选项，但LabVIEW仍禁用了某些复选框。这是因为尚未在**寄存器**页面添加相应的寄存器。单击**寄存器**选项卡，勾选相应的复选框。然后返回**启用**或**重置**页面。

单击**确定**按钮并返回程序框图时，LabVIEW显示您选中的任意布尔输入接线端。然后可连线代码，以将这些接线端在适当的时间设置为TRUE或FALSE。

等效于LabVIEW选项的VHDL代码

如要查看配置的相应VHDL代码，单击**VHDL实例**选项卡。关于指定属性的详细信息，见***Virtex-5 FPGA XtremeDSP Design Considerations User Guide***或***Virtex-6 FPGA DSP48E1 Slice User Guide***中的表1-3。通过Xilinx网站`www.xilinx.com`可获取该文档。下表为对应于LabVIEW中每个选项的属性。

LabVIEW选项	表1-3的参考属性
	AREG
	BREG
	ACASCREG
	BCASCREG
alumode寄存器	ALUMODEREG
c寄存器	CREG
carryin寄存器	CARRYINREG
carryinsel寄存器	CARRYINSELREG
(DSP48E) **m寄存器**	MREG
(DSP48E) **multcarryin寄存器**	MULTCARRYINREG
(DSP48E1) **m寄存器和multcarryin寄存器**	MREG
opmode寄存器	OPMODEREG
p寄存器	PREG
(DSP48E1) **d寄存器**	DREG
(DSP48E1) **ad寄存器**	ADREG
(DSP48E1) **inmode寄存器**	INMODEREG

相关概念：

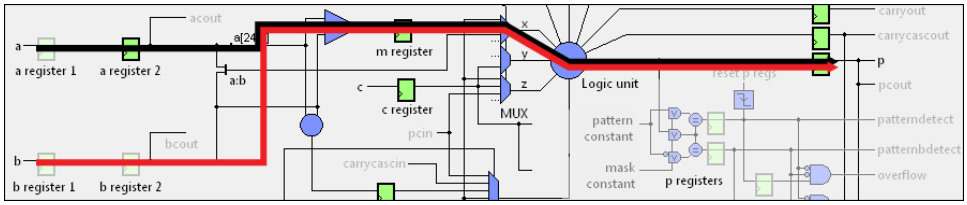
- 查看DSP48E和DSP48E1函数的数据路径延迟
- 显示和隐藏DSP48E或DSP48E1函数的接线端
- 配置DSP48E和DSP48E1函数

查看DSP48E和DSP48E1函数的数据路径延迟

使用配置对话框底部的示意图查看DSP48E或DSP48E1函数内部的数据路径延迟。这些延迟为数据从输入接线端流动至输出接线端所需的时钟周期数。例如，**a**到**p**为1个数据路径，**b**到**p**为另一个数据路径。在数据路径中每个添加的寄存器会为该数据路径增加一个时钟周期，但不会影响其他数据路径的延迟。因此，数据路径延迟在配置函数时会出现不平衡。

该不平衡是否可被接受取决于用户使用函数的方式。某些应用不需要具有平衡数据延迟的数据路径。例如，如果连线一个函数至另一个函数，每个函数内部的不平衡可被其他函数内的相等但相反的路径抵消。但某些应用需要具有平衡数据延迟的数据路径。下列范例显示了需要平衡的数据路径的情况。

假设函数被配置为乘法器，**p** = **a** ^{2}。也就是连线相同的值到函数的**a**和**b**输入接线端。该值由0开始，每个时钟周期增加1。DSP48E函数包含下列数据路径：



在该示意图中，**a**–>**p**数据路径包含下列寄存器：

- **a寄存器2**
- **m register**
- **p**

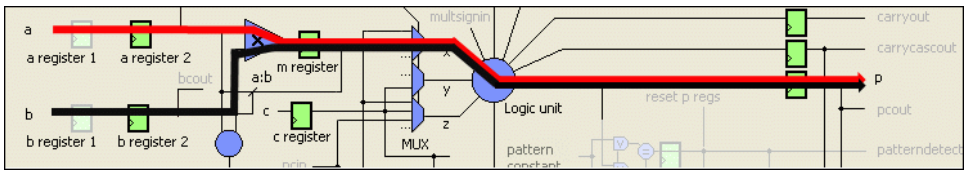
因为该数据路径包含3个寄存器，所以数据路径的延迟为3个时钟周期。但**b**–>**p**数据路径仅包含2个寄存器，因此延迟为2个时钟周期。延迟不平衡意味着：

- 对于**a**–>**p**数据路径，第n个时钟周期使用第(n – 3)个时钟周期的值**a**
- 对于**b**–>**p**数据路径，第n个时钟周期使用第(n – 2)个时钟周期的值**a**

由于该不平衡，无法实现计算**p** = **a** ^{2}的操作。下表显示了函数的前8个时钟周期内的输入和输出：

时钟周期	a的值	P的值	说明
1	0	N/A	在此时钟周期，没有数据路径产生值。
2	1	N/A	
3	2	0	函数乘以第二个寄存器的初始化值和0（来自第一个时钟周期的**a**值）。
4	3	0	函数乘以0（来自第一个时钟周期的**a**值）和1（来自第二个时钟周期的**a**值）。
5	4	2	函数乘以1（来自第二个时钟周期的**a**值）和2（来自第三个时钟周期的**a**值）。
6	5	6	函数乘以2（来自第三个时钟周期的**a**值）和3（来自第四个时钟周期的**a**值）。
7	6	12	函数乘以3（来自第四个时钟周期的**a**值）和4（来自第五个时钟周期的**a**值）。
8	7	20	函数乘以4（来自第五个时钟周期的**a**值）和5（来自第六个时钟周期的**a**值）。

在该应用中，延迟的不平衡将导致错误的计算结果。此时，通过**寄存器**页面添加另一个寄存器至**b**–>**p**数据路径。此时示意图显示如下：



在上图中，请注意2个数据路径均带有3个寄存器。即相对于双方数据路径，第n个时钟周期均使用(n – 3)个时钟周期的**a**值。

下表显示了函数的前8个时钟周期内的输入和输出：

时钟周期	a的值	P的值	说明
1	0	N/A	在此时钟周期，没有数据路径产生值。
2	1	N/A	
3	2	N/A	
4	3	0	函数乘以0和0（来自第一个时钟周期的a值）
5	4	1	函数乘以1和1（来自第二个时钟周期的a值）
6	5	4	函数乘以2和2（来自第三个时钟周期的a值）
7	6	9	函数乘以3和3（来自第四个时钟周期的a值）
8	7	16	函数乘以4和4（来自第五个时钟周期的a值）

当前数据路径的延迟是平衡的。DSP48E函数能够正确计算**p** = **a** ^{2}。

相关概念：

- 调整DSP48E或DSP48E1函数的内部组合路径的长度
- 配置DSP48E和DSP48E1函数
- 保持DSP48E和DSP48E1函数接线端的精度

保持DSP48E和DSP48E1函数接线端的精度

如要保持DSP48E或DSP48E1函数的数值精度，请使用配置对话框的**定点数配置**页面正确设置输入和输出接线端的整数字节长度。LabVIEW将上述接线端的字节长度保持为常量，即整数字节长度为用户可控制的变量。

LabVIEW将特定的函数接线端组合在一起。这些组必须保持相同的定点数数据类型。下表显示了每个组的接线端和字节长度。

组内的接线端	接线端的字节长度
a, **acin**	当在**函数**页面上选择了**算术配置**并使用乘法器时，字长在直接模式下为25，在级联模式下为30。否则，字长为30。下列情况除外： • 在UltraScale终端上，只有低27位送入乘法器。 • 在非UltraScale终端上，只有低25位送入乘法器。
acout	30
b、**bcin**和**bcout**	18
c	48
p、**pcin**和**pcout**	48
(DSP48E1) **d**	25

注： 关于DSP48E和DSP48E1函数的输入和输出接线端的详细信息，见Virtex-5 FPGA XtremeDSP Design Considerations User Guide的"A, B, and C ports"章节，或Virtex-6 FPGA DSP48E1 Slice User Guide的"A, B, C and D ports"章节。通过Xilinx网站www.xilinx.com可获取该文档。

保持a和b接线端的精度

建议按照下列规范设置**a**和**b**接线端的整数字节长度。

- 上述接线端的小数字节长度等于字节长度减去整数字节长度，其长度大于或等于连线至接线端的数据类型的小数字节长度。
- 接线端的整数字节长度大于或等于源的整数字节长度。

例如，连线<+/–,16,1>数据类型至**b**接线端，数据类型的小数字节长度为15位。由于**b**的字节长度为固定的18位，设置**b**的整数字节长度为3、2或1位。设置可满足上述

准则：小数字节长度为15、16或17位。**b**的整数字节长度大于或等于源的整数字节长度。

如整数字节长度小于1位或大于3位，LabVIEW将强制转换该值至**b**接线端，从而导致精度的损失。在输入接线端将出现强制转换点，以提醒用户该情况。

保持乘法器的精度

如要保持配置为**p** = **a** * **b**的乘法器的DSP48E或DSP48E1函数的全部精度，设置**p**的小数字节长度为**a**和**b**接线端的小数字节长度的和。

请参考以下范例：

- 源至**a**的数据类型为<+/–,16,1>。小数字节长度为15位。
- 源至**b**的数据类型为<+/–,16,2>。小数字节长度为14位。

小数字节长度的和值为29位。在该情况下建议执行下列步骤：

1. 设置**a**为<+/–,30,15>。该设置与连接至该接线端的15位小数字节长度匹配。
2. 设置**b**为<+/–,18,4>。该设置与连接至该接线端的14位小数字节长度匹配。
3. 设置**p**为<+/–,48,19>。该设置与**a**和**b**接线端组合的29位小数字节长度匹配。

保持累加器的精度

如要保持配置为**p** = **p** + **c**的累加器的DSP48E或DSP48E1函数的全部精度，设置**p**的小数字节长度与**c**的小数字节长度相等。例如，如源至**c**的数据类型为<+/–,16,3>，**c**的小数字节长度为13位。设置**p**为<+/–,48,35>。

相关概念：

- 查看DSP48E和DSP48E1函数的数据路径延迟
- 显示和隐藏DSP48E或DSP48E1函数的接线端

在DSP48E和DSP48E1函数间切换

DSP48E函数和DSP48E1函数可进行相互替换，具体方法为右键单击函数，分别选择**替换为DSP48E1**或**替换为DSP48E**。例如，从Virtex-5 FPGA终端移动VI至Virtex-6 FPGA 终端时，可能需要替换DSP48E函数为DSP48E1函数。

使用DSP48E和DSP48E1函数的说明

下表列出了在FPGA终端上编译DSP48E或DSP48E1函数导出用于仿真的函数的区别。

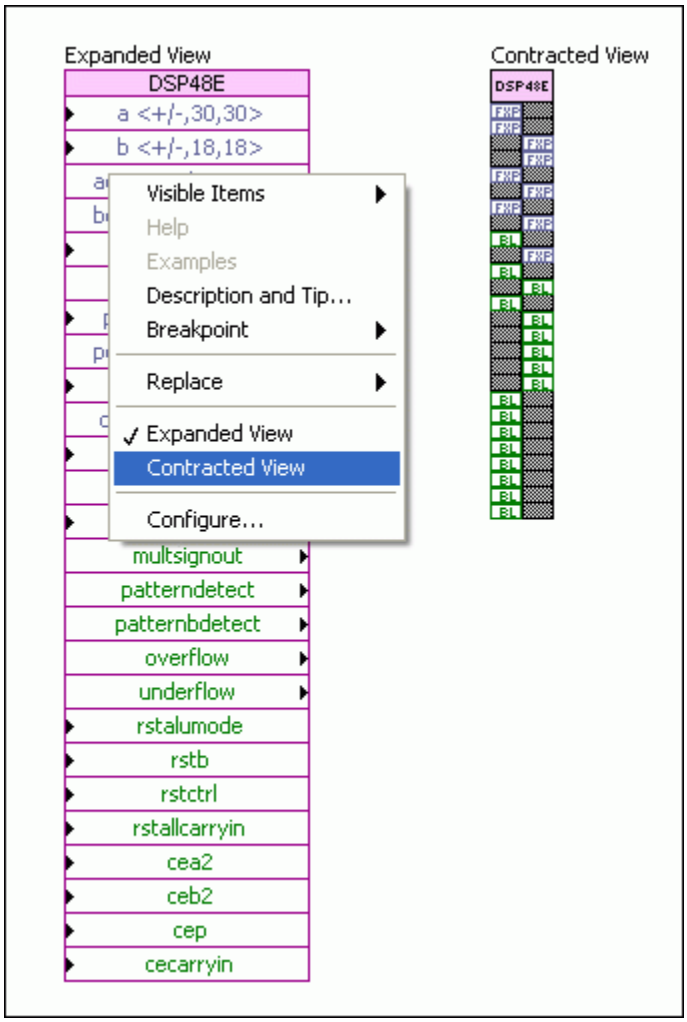
	在FPGA终端执行	**仿真**
支持的执行模式	单周期定时循环内部	• 单周期定时循环内部 • 单周期定时循环外部
可选接线端的连线限制：	• 仅可连接一个函数的**acout**输出端至另一个函数的**acin**输入端 • 仅可连接一个函数的**bcout**输出端至另一个函数的**bcin**输入端 • 仅可连接一个函数的**pcout**输出端至另一个函数的**pcin**输入端 • 仅可连接一个函数的**multsignout**输出端至另一个函数的**multsignin**输入端 • 仅可连接一个函数的**carrycascout**输出端至另一个函数的**carrycascin**输入端 • 不能创建上述连线的分支、创建上述输出接线端的显示控件或添加探针。	可创建上述连线的分支、创建上述输出接线端的显示控件或添加探针。

相关概念：

- 使用第三方仿真器调试FPGA VI
- 显示和隐藏DSP48E或DSP48E1函数的接线端
- 配置DSP48E和DSP48E1函数

缩放DSP48E或DSP48E1函数按钮

如要减少DSP48E或DSP48E1函数图标在程序框图上所占的空间，右键单击函数并选择**折叠视图**。下图显示了两种不同视图的区别：



相关概念：

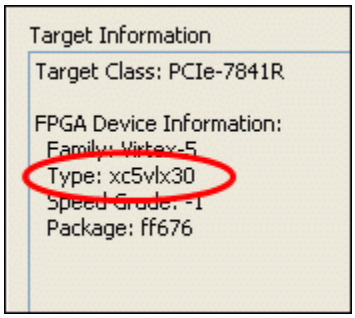
- DSP48E范例：创建复数乘法器
- DSP48E范例：创建一个n阶FIR滤波器

确定DSP48E或DSP48E1函数的最大可用数量

每个添加至程序框图的DSP48E或DSP48E1函数代表一个DSP48E或DSP48E1逻辑片。每个FPGA的可用DSP48E或DSP48E1逻辑片的数量是确定的。按照下列步骤判定可用的函数的最大数量。

注： DSP48E函数可用于Virtex-5和Virtex-6FPGA终端；但DSP48E1仅可用于Virtex-6终端。在Virtex-6 FPGA终端上，DSP48E函数使用一个DSP48E1逻辑片。

1. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**属性**。LabVIEW打开FPGA终端属性对话框。
2. 查看**终端信息**文本框，了解FPGA的**类型**。**xc5v**的**类型**指示终端为Virtex-5 FPGA。**xc6v**的**类型**指示终端为Virtex-6 FPGA。例如，NI PCIe-7841R的**类型**为**xc5vlx30**，表明其为Virtex-5 FPGA且仅支持DSP48E函数：



3. DSP48E：见Virtex-5 FPGA XtremeDSP设计考虑要素用户指南中表1-1。通过Xilinx网站www.xilinx.com可获取该文档。DSP48E1：见***Virtex-6 FPGA DSP48E1逻辑片用户指南***中的表1-1。
4. DSP48E：查找与FPGA**类型**匹配的**设备DSP**表格。下一栏给出了该类型的FPGA的最大可用DSP48E逻辑单元的数量。例如，xc5vlx30类型具有32个DSP48E逻辑片。该数量表示如对NI PCIe-7841R进行编程，可添加最多32个DSP48E函数至程序框图。(DSP48E1)查找与FPGA的**类型**匹配的**全部DSP48E1逻辑片/设备**表格。该列显示了可添加至程序框图的DSP48E1和DSP48E函数的最大数量。

相关概念：

DSP48E范例：创建复数乘法器

用户可使用4个DSP48E逻辑单元实现复数乘法器。乘法器的输入为4个定点数（x1、x2、y1和y2），每对表示一个复数：

- x1 + j * y1
- x2 + j * y2

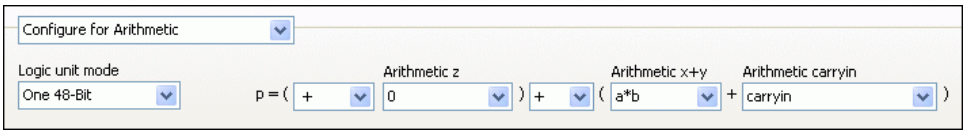
复数乘法器使用这些输入并生成2个定点数：

- x1 * x2 – y1 * y2
- x1 * y2 + x2 * y1

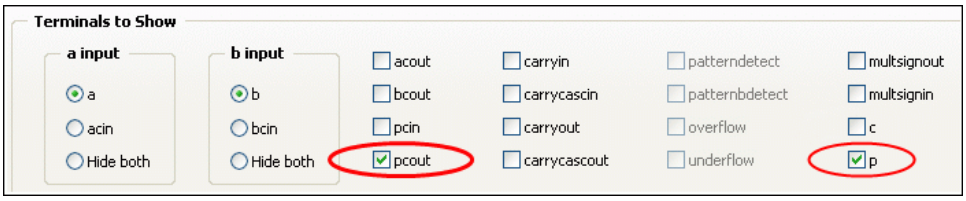
然后可计算 (x1 * x2 – y1* y2) + j(x1 * y2 + x2 * y1).

按照下列步骤创建一个复数乘法器。

1. 在FPGA终端上新建一个VI。
2. 添加DSP48E函数至程序框图。
3. 双击函数并按照下列方式配置函数：
   a. **函数**页面：



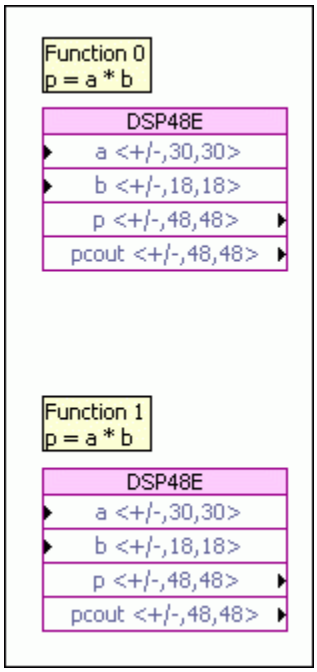
   b. **接线端**页面：



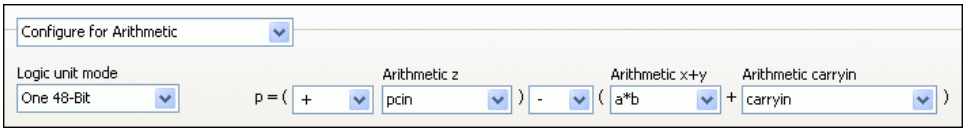
   c. 单击**确定**按钮保存更改，并返回程序框图。

此函数当前为函数0。

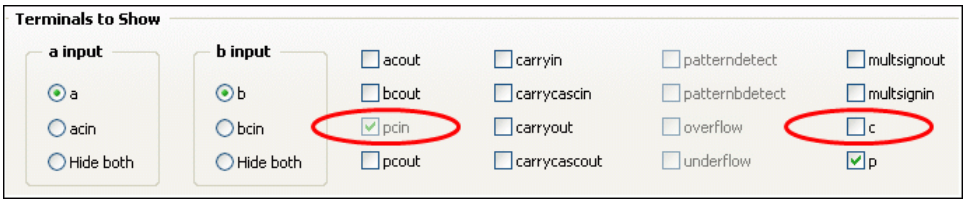
4. 按下<Ctrl>键，然后拖曳函数至当前函数的下方。此操作将创建DSP48E函数的副本－函数1。此副本与之前的函数具有相同的配置。当前的程序框图将显示下图：



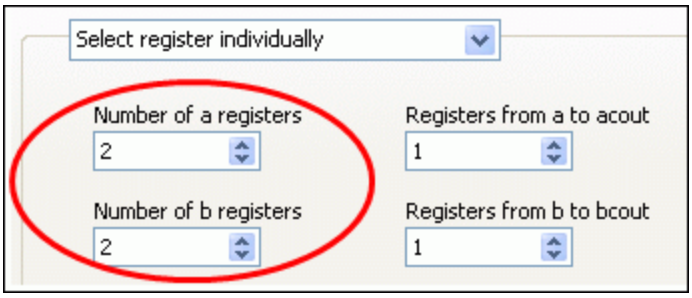
5. 添加另一个DSP48E函数至函数0的右侧。该函数为函数2。
6. 双击函数并按照下列方式配置函数：
   a. **函数**页面：



   b. **接线端**页面：



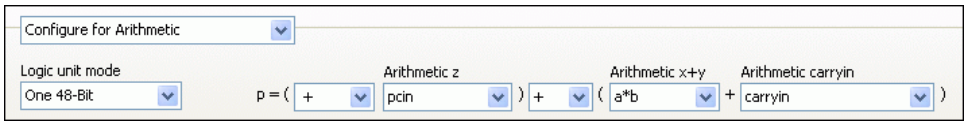
   c. **寄存器**页面：



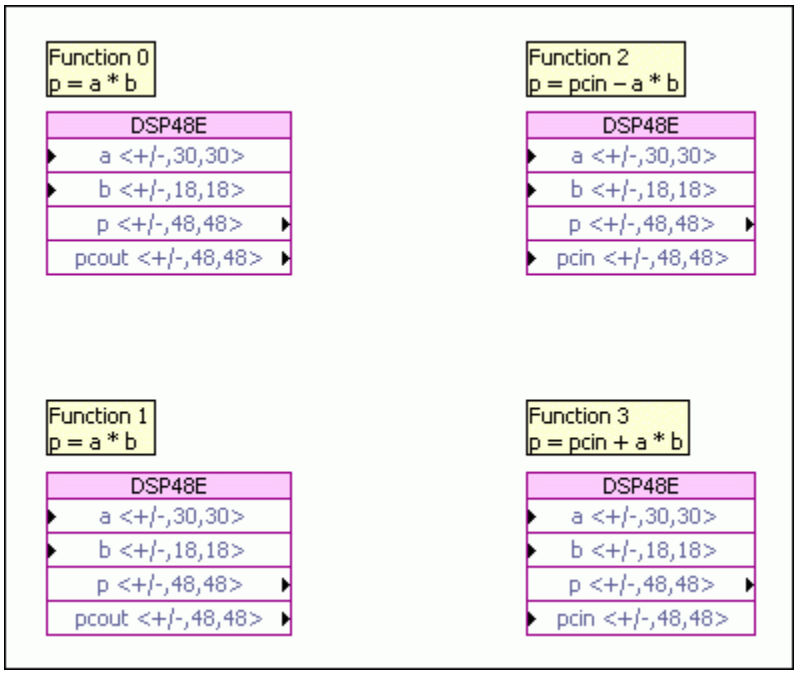
添加额外的电阻至每个数据路径，能够确保此应用中的延时平衡。

d. 单击**确定**按钮保存更改，并返回程序框图。

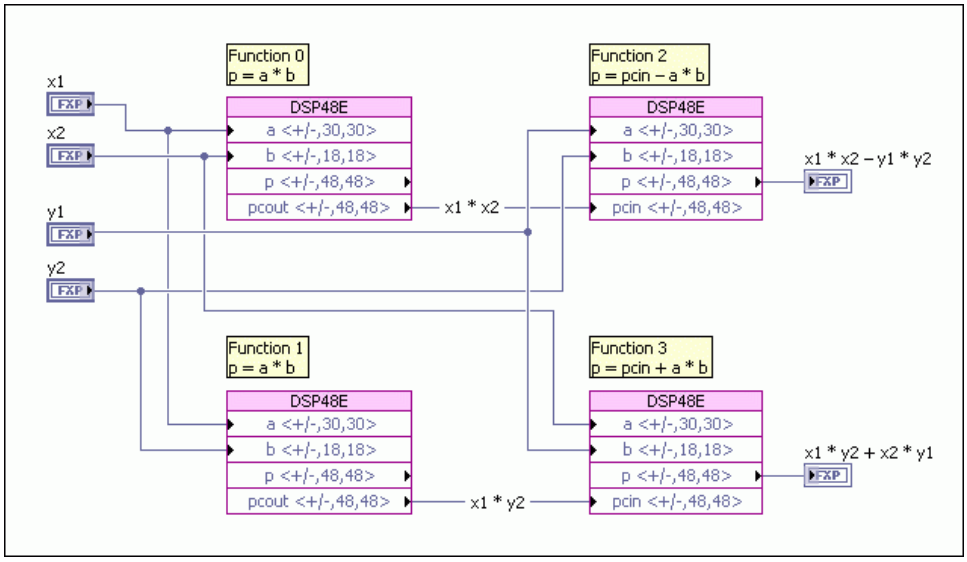
7. 按下<Ctrl>键，拖曳函数至函数1的右侧，函数2的下方。
8. 双击此副本，并按照下列方式配置**函数**页面。



9. 单击**确定**按钮保存更改并返回程序框图。程序框图将显示下图：



10. 按照下图创建并连线输入控件和显示控件：



注： 根据x1、x2、y1和y2的数据源，可能需要配置DSP48E输入接线端的整数字节长度。

11. 保存VI至可用的路径为`DSP48E Complex Multiplier.vi`。

此VI可被输出以用于仿真。如要在FPGA终端上运行此VI，必须在单周期定时循环中关闭此代码。

提示 在程序框图上可缩放函数按钮以节省空间。

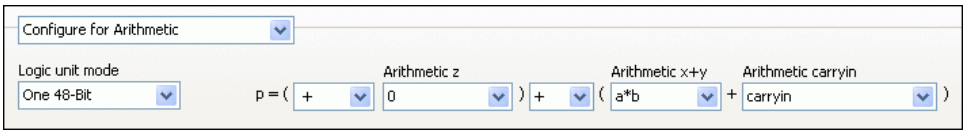
相关概念：

- 配置DSP48E和DSP48E1函数
- 确定DSP48E或DSP48E1函数的最大可用数量
- 缩放DSP48E或DSP48E1函数按钮

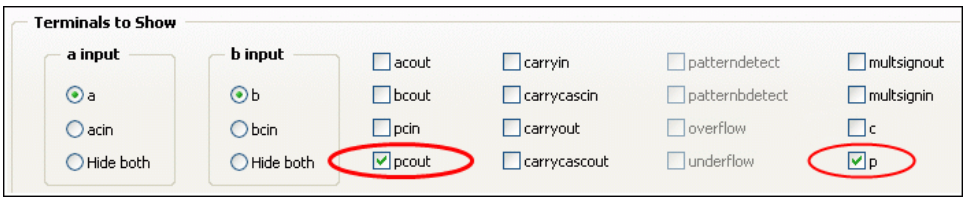
DSP48E范例：创建一个n阶FIR滤波器

使用n个DSP48E函数可创建一个n阶FIR滤波器。每个函数处理FIR滤波器的一个系数或一阶。其中h[0..n–1]表示FIR滤波器的n个系数。按照下列步骤，创建一个3阶FIR滤波器。

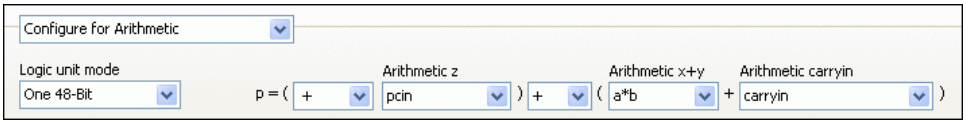
1. 在FPGA终端上新建一个VI。
2. 添加DSP48E函数至程序框图。
3. 双击函数并按照下列方式配置函数：
   a. **函数**页面：



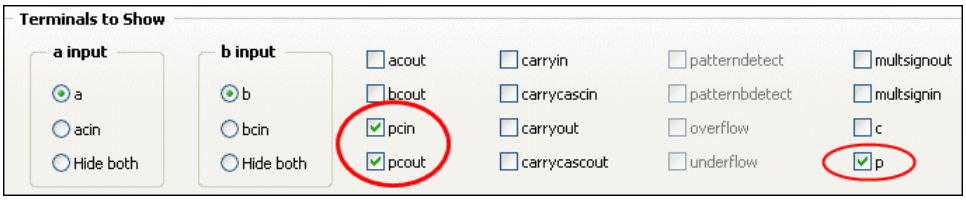
   b. **接线端**页面：



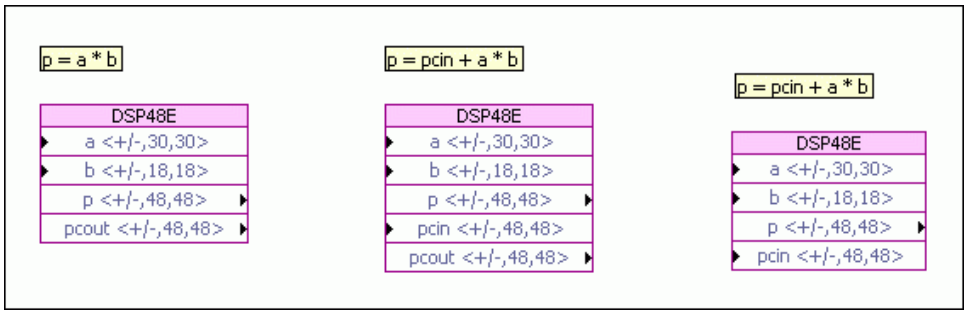
   c. 单击**确定**按钮保存更改，并返回程序框图。
4. 添加另一个DSP48E函数至程序框图。按照下列方式配置函数：
   a. **函数**页面：



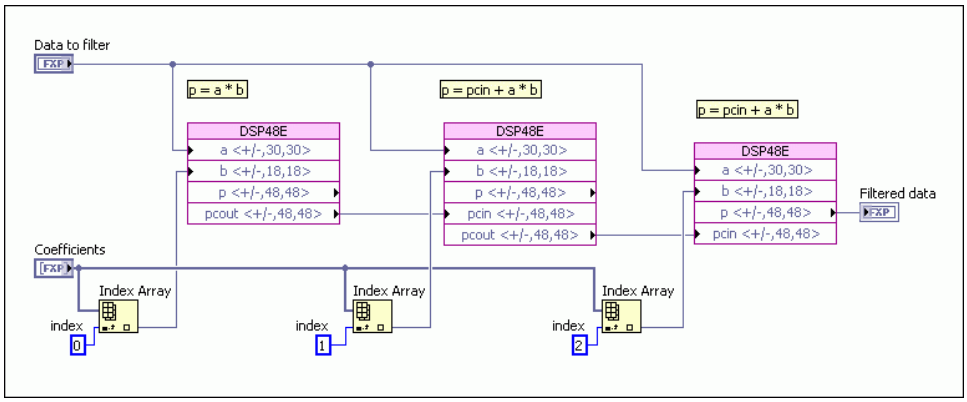
   b. **接线端**页面：



   c. 单击**确定**按钮保存更改，并返回程序框图。
5. 按下<Ctrl>键，在程序框图上拖曳此节点以创建节点的副本。重复此步骤，直至程序框图上有总数为n个DSP48E函数。在此3阶滤波器范例中，程序框图上共有3个DSP48E函数。
6. 双击第n个函数，单击**接线端**选项卡，取消勾选**pcout**复选框。
7. 单击**确定**按钮保存更改，并返回程序框图。



8. 连线函数和接线端。
   a. 连线至滤波器的数据至每个DSP48E函数的**a**输入端。根据该输入的源，可能需要调整**a**输入端的整数字节长度。
   b. 连线每个滤波器的系数至每个后续DSP48E函数的**b**输入端。可能需要缩放连线至这些接线端的值。
   c. 连线每个**pcout**输出接线端至下一个DSP48E函数的**pcin**输入端。
   d. 最后一个DSP48E函数的**p**输出接线端返回滤波后的数据。或者为此函数创建一个显示控件，或连线其至另一个函数。
9. 当前的程序框图将显示下图：



10. 保存VI至可用的路径为`DSP48E FIR Filter.vi`。

在上图中，每个函数的**a**输入端处理想要滤波的数据。**b**输入端处理滤波器系数h[0]至h[n–1]。共有n个系数，每个DSP48E函数带有一个系数。

提示 如n较大，建议采取下列建议：

- 大型数组将占用大量的FPGA空间。因此建议使用另一种方法存储系数，如存储器项。或者使用“离散延迟”存储用于一系列时钟周期的多个系数。
- 程序框图上将存在多个DSP48E按钮。缩放这些按钮能够减少每个按钮的使用空间。

此VI可被输出以用于仿真。如要在FPGA终端上运行此VI，必须在单周期定时循环中关闭此代码。

相关概念：

- 配置DSP48E和DSP48E1函数
- 缩放DSP48E或DSP48E1函数按钮

存储和传输数据

设计存储或传输数据的FPGA引用时，用户需指定应用用于处理数据的FPGA资源大小和类型。如应用所需的FPGA资源超出设备的可用资源，则不会执行编译。

如要创建一个可行的FPGA设计，请选择与所需执行任务匹配的FPGA资源类型和大小。

下列主题提供了选择用于应用的最佳设计组件的详细信息。

- 在FPGA终端上存储数据
- 在架构或设备间传输数据
- 配合使用点对点数据流和FPGA终端

相关概念：

- 在FPGA终端上存储数据
- 在设备或FIFO架构间使用FIFO传输数据

使用带有寄存器项、存储器项、FIFO和握手项的自定义数据类型

创建支持数据类型的自定义控件，并选择上述控件为用于FPGA寄存器项、存储器项、FIFO和握手项的数据类型。在寄存器属性、存储器属性、FIFO属性或握手属性对话框中选择用作数据类型的自定义控件可以是通过支持的数据类型创建的任意控件、自定义类型(typedef)或严格自定义类型控件。如自定义控件为自定义类型或严格自定义类型，LabVIEW将断开控件与自定义类型的连接。

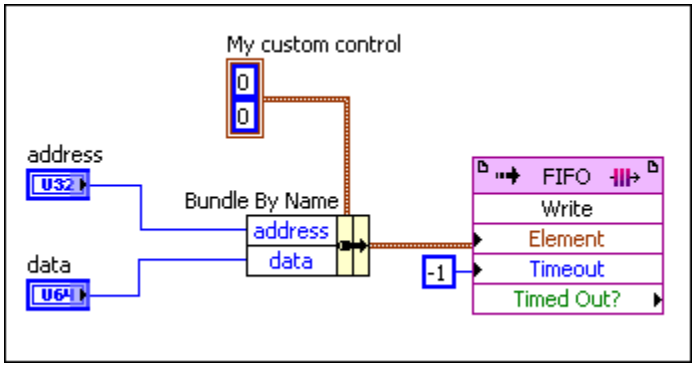
使用自定义数据类型具有下列优点：

- 简化程序框图可增加应用吞吐率。
- 对于某些FPGA终端，合并两个小的支持数据类型为较大的数据类型打破了原有的64位限制。关于支持的数据类型的详细信息，见指定FPGA终端的硬件文档。
- 使每个值在FIFO或存储器中的资源独立。

注： DMA和点对点的FIFO不支持自定义数据类型。

保持资源属性独立的同时，组合较小的数据类型

在下列范例中，FIFO可在单个簇元素中转换地址和数据信息。通过使用自定义控件，这些值与其他值区别开来，以便从FIFO中读取元素后拆分这些值。



使用自定义控件的说明

使用大的自定义控件将影响FPGA VI的性能或耗费FPGA资源。

相关概念：

- 优化FPGA VI的执行速度和大小
- 支持的数据类型

通过主控计算机与FPGA终端通信

通过下列方式与FPGA终端通信：

- 交互式前面板通信—FPGA终端执行程序框图时，在主控计算机上显示FPGA VI的前面板窗口。
- 可编程式FPGA接口通信—使用运行在外部计算机上的独立VI控制、监控FPGA终端及传输数据。用于编程控制和监控FPGA VI的VI称为主控VI，运行主控VI的机器称为主控计算机。
- 点对点数据流—在两个硬件设备间传输数据。

下表总结并比较了上述方法。

通信方法	主控OS	控制和监控机制	常用于	说明
交互式前面板通信	Windows	FPGA VI的前面板	• 简单逻辑验证 • 简单调试	• 并非支持所有终端 • 无需创建运行在主控计算机上的VI。 • 不支持使用DMA FIFO或FPGA终端通信。
可编程式FPGA接口通信	Windows, RT	运行在主机上的VI	• 在主机上记录数据 • 集成FPGA、RT和桌面组件至一个应用 • FPGA上不能执行的操作 • 测试台	• 最常见的编程方法 • 支持可编程式前面板通信

通信方法	主控OS	控制和监控机制	常用于	说明
			• 与远程FPGA终端交互 • 使用DMA FIFO或FPGA中断的应用	
点对点数据流	Windows, RT	运行在主机上的VI	• 两个FPGA终端间的数据流数据，未进入主机处理器 • 多个数字化仪至FPGA终端的数据流 • 将数字化仪至FPGA终端的数据并行至信号发生器	• 并非支持所有终端 • 需要两个终端及主控计算机

在可能的情况下，NI建议使用可编程FPGA接口通信和主控VI。使用交互式前面板与FPGA VI通信的优势在于无需编程额外的VI，但使用交互式前面板通信没有使用可编程FPGA交互式通信功能强大。

同步使用交互式和程序控制通信

虽然可同时使用交互式前面板通信和程序控制FPGA接口通信，但建议每次仅使用一个通信选项控制和监控FPGA VI。如要同时使用两种通信选项有效的调试FPGA VI，可考虑通过主控VI控制数据的读写，通过交互式前面板通信监控执行。

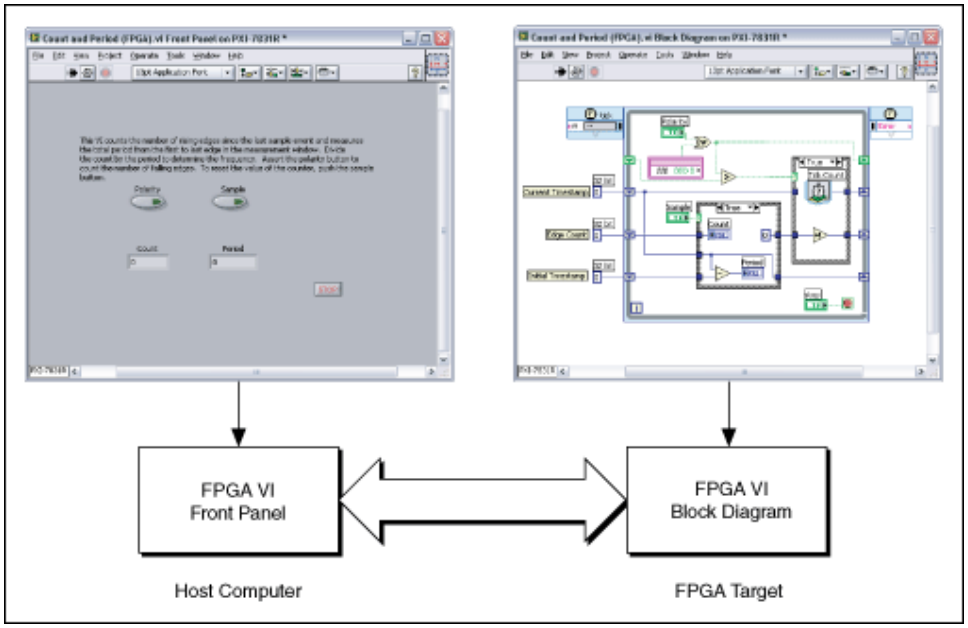
相关概念：

- 交互式前面板通信
- 使用主控VI与FPGA终端通信
- FPGA编程概述

交互式前面板通信

通过交互式前面板通信与运行在FPGA终端上的FPGA VI通信，无需额外的编程操

作。在交互式前面板通信中，主控计算机显示FPGA VI的前面板窗口，FPGA终端执行FPGA VI程序框图，如下列示意图所示。



注： 交互式前面板通信的支持随FPGA终端改变。更多信息见FPGA终端硬件的文档。

LabVIEW前面板窗口通过输入控件和显示控件与FPGA终端程序框图通信。可与直接连接至主控计算机或通过网络连接至远程系统的FPGA终端通信。FPGA终端程序框图继续运行时，主控计算机将尽可能地更新FPGA VI前面板窗口的值。FPGA VI的执行速率不受主控计算机通信的影响。但通过交互式前面板通信共享的前面板数据具有不确定性。

通过FPGA终端和主控计算机间的交互式前面板通信控制和测试运行在FPGA终端的VI。下载和运行FPGA VI后，保持主控计算机上的LabVIEW打开，以显示FPGA VI的前面板窗口并进行交互。

在交互式前面板通信中，不能使用LabVIEW调试工具，包括探针、高亮显示执行过程、断点及单步执行。如要在FPGA终端编译、下载及运行FPGA VI前发现错误，考虑使用测试台。

提示FPGA配置为在第三方仿真器上运行时，不能使用交互式前面板通信。在“项目浏览器”窗口右键单击FPGA终端，并选择**选择执行模式**，可使用主控VI执行FPGA VI，或更改FPGA终端的执行模式。

相关概念：

- 编译、下载和运行FPGA VI
- 通过主控计算机与FPGA终端通信
- 创建仿真I/O的自定义VI

使用变量存储数据

局部或全局变量可用于在FPGA的应用中存储数据。变量仅存储写入其内的最新的值。如读取变量值前写入了N次，则最后一个值前的N-1个值均丢失。如不需要使用每个获取的数据值，变量是一个良好选择。因为无需编写其他代码以丢弃不需要的值。

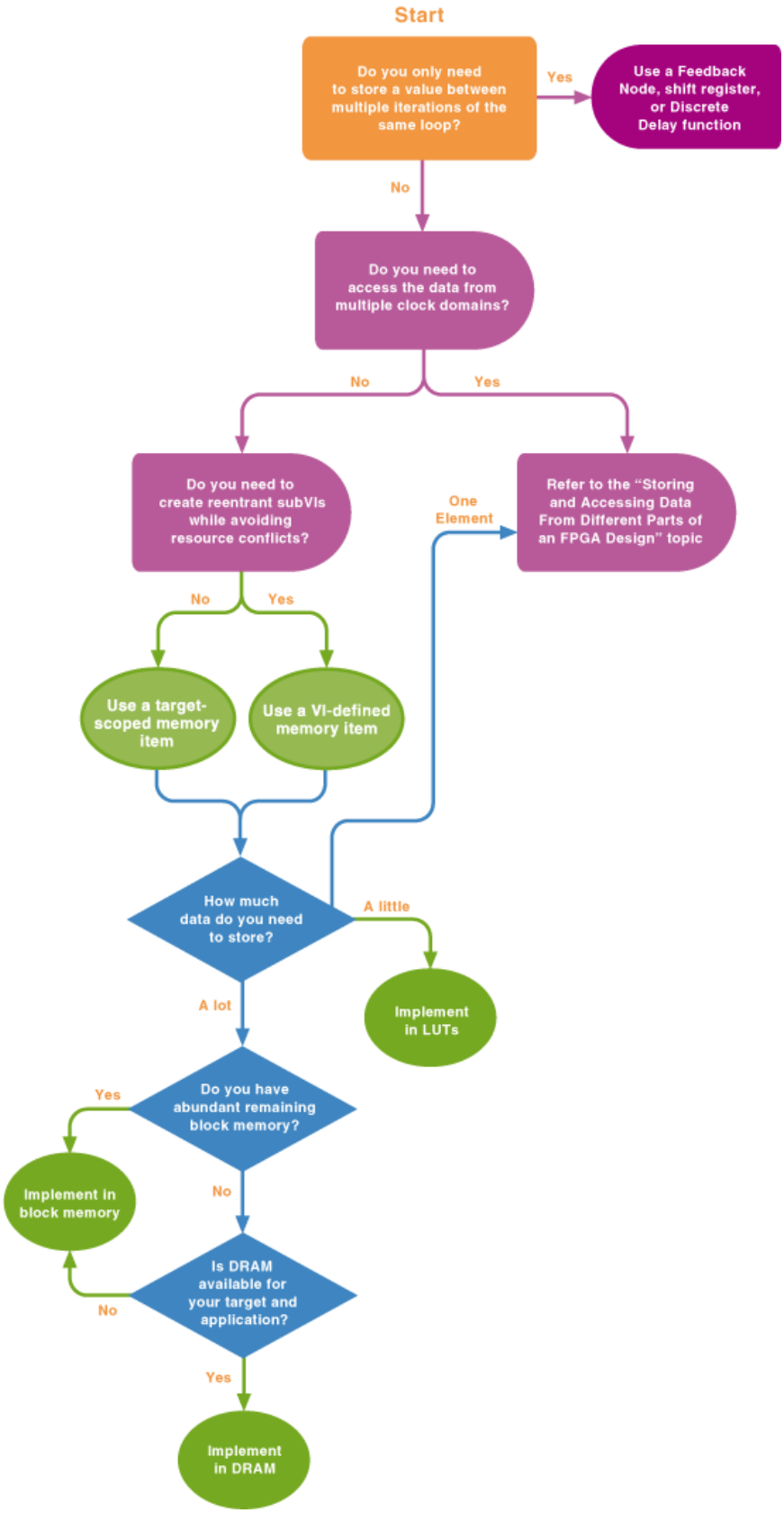
全局变量位于项目浏览器窗口的FPGA终端下，通过运行在FPGA终端上的VI可访问它们。使用全局变量存储要从多个VI访问的数据。使用局部变量存储仅需通过单个VI访问的数据，或需要独立维护的用于重入子VI的独立实例。仅可通过单个VI访问局部变量。但当在重入子VI中包含一个局部变量时，子VI的每个实例均包含局部变量的一个新的实例。

注： 如包含对同一变量的多次读/写，变量可用作共享资源。如要防止数据损坏和抖动，应避免同步发送读取或写入请求至单个变量。

在FPGA终端上存储数据

- 寄存器项
- 存储器项
- 反馈节点或移位寄存器
- 全局和局部变量

- 离散延迟函数
- 数组


Start
Do you only need to store a value between multiple iterations of the same loop?
Yes
Use a Feedback Node, shift register, or Discrete Delay function
No
Do you need to access the data from multiple clock domains?
No
Yes
Do you need to create reentrant subVIs while avoiding resource conflicts?
Refer to the "Storing and Accessing Data From Different Parts of an FPGA Design" topic
One Element
No
Yes
Use a target-scoped memory item
Use a VI-defined memory item
How much data do you need to store?
A little
Implement in LUTs
A lot
Do you have abundant remaining block memory?
Yes
Implement in block memory
No
Is DRAM available for your target and application?
No
Yes
Implement in DRAM

注： 通过FPGA终端属性对话框的常规页面的**终端信息**组件，获取FPGA终端支持的内存块的数量信息。

相关概念：

- 存储和传输数据
- 通过双端口读取降低存储器资源使用

FPGA存储器项

单时钟域内使用FPGA应用的主要数据存储方式为存储器项。

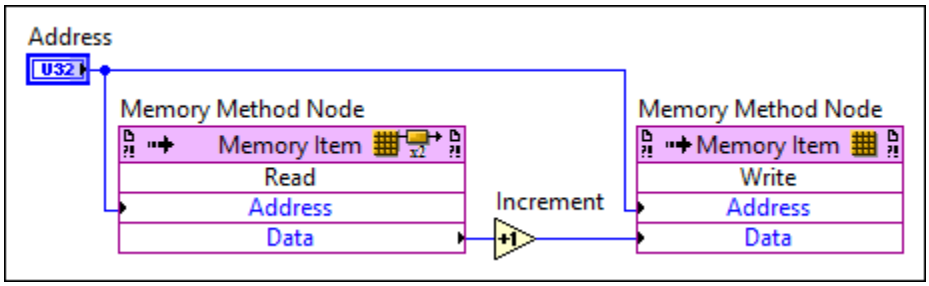
LabVIEW FPGA模块具有两种类型的存储器项：

- **VI定义的存储器项**：使用VI定义的存储器项和存储器名称控件创建可重用子VI并避免资源冲突。在可重入FPGA VI中使用VI定义的存储器项时，LabVIEW为每个VI的实例创建独立的FIFO的副本。
- **终端范围的存储器项**：如需存储器项可见并可通过**项目浏览器**窗口配置，使用终端范围的存储器项。终端范围的存储器项按名称被静态捆绑至项目中的相应项，意味着更新项目的项将影响程序框图中存储器项的全部实例。终端范围的存储器项在**项目浏览器**窗口中同一终端下的任意FPGA VI内均可用。如使用终端范围的存储器项并想要将其发送给另一用户，必须发送整个项目。否则FPGA VI将显示为断开。

注： 如包含多个对同一存储器项的读取或写入，存储器项可用作共享资源。如要防止数据丢失和抖动，应避免同步发送读取或写入请求至单个存储器项。

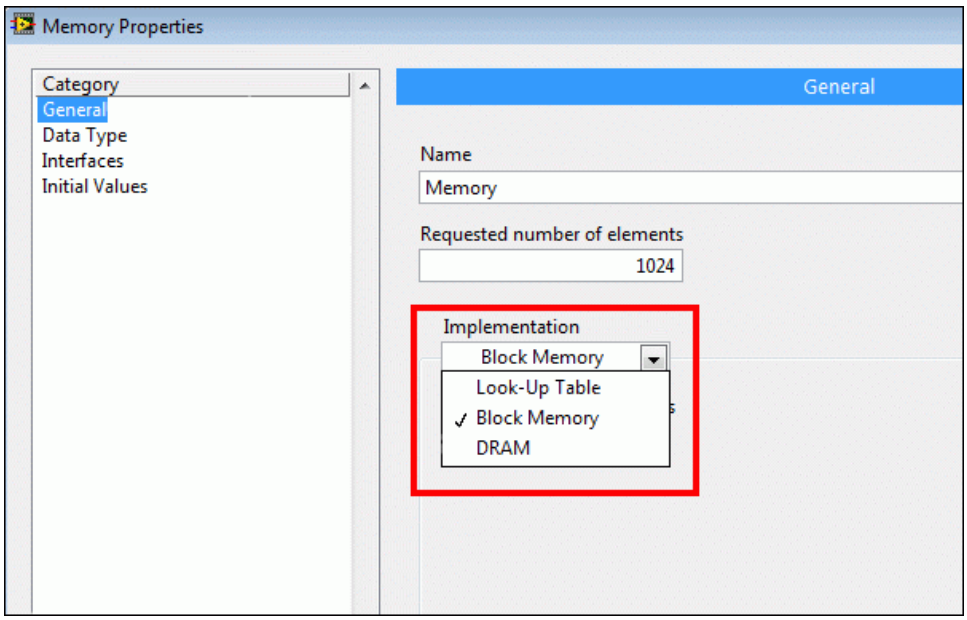
下列程序框图为使用存储器方法节点读取和写入存储器项的方法，该节点配置用于终端范围的存储器项。该VI由内存读取数据、递增数据并使用新数据覆盖同一内存位置的值。

注： 仅当位于单周期定时循环外时，该程序框图有效。如在单周期定时循环内使用该程序框图，必须连线至少两个反馈节点或未初始化的移位寄存器至读取（存储器方法）的**数据**输出端，因为存储器项的读取延迟周期置为2。读取（存储器方法）节点图标下的x2指示了存储器项的延迟周期数。



存储器实现选项

使用**存储器属性**对话框指定实现存储器项的方式。展开**执行**下拉菜单显示可用的存储器项，如下列示意图所示。



注： 关于配置该对话框中其他选项的帮助信息，见**存储器属性**对话框。

存储器块

存储器块（通常也被称为块随机访问内存、块RAM或BRAM）是用于数据存储的内部FPGA资源。使用存储器块的存储器项相对于其他存储器项采用高速时钟速率编译。存储器块可被配置为读写访问或双端口读取访问。或者使用通过存储器块实现的存储器项在一个时钟域内写入数据，从另一个时钟域读取数据。在上述操作中，每个存储器项仅使用一个写入方节点和一个读取方节点。存储器块不占用FPGA资源。除非需要使用其他类型存储器的特有特性，否则请优先考虑使用存储器块。

警告 当在多时钟域使用块内存实现存储器项时, 可同时从同一地址读取和写入数据。但可能导致读取不正确的数据。

读取延迟周期的值增加将增加内部流水线层级，同时也会增加编译设计的最大频率。可在**存储器属性**对话框的**常规**页面，为使用存储器块实现的存储器项指定读取延迟周期的值。请参考下列表格，判定设计使用的读取延迟周期的数量。

读取延迟周期	推荐使用	最大频率的影响
0	使用查找表实现存储器时可用	无更改
1	如需要尽可能快的获取有效数据，但没有足够的FPGA资源以使用查找表，此时选择该设置	无更改
2（默认）	多数应用的最大频率的推荐设置	增加
3	需要存储大量数据时的推荐设置	增加
	 **提示** 如Xilinx编译报告显示应用程序使用超过一个存储器块，使用该设置。	

LUT（查找表）

LUT（即分布式RAM）由FPGA上硬连线的逻辑门组成。由于LUT可用作逻辑资源或存储器，因为LUT将消耗FPGA资源。在下列情况下使用查找表：

- 在单周期定时循环中访问该存储器，且需要在用户调用节点的同一时钟周期读取存储器项的数据。
- 可用存储器块数量有限

动态RAM (DRAM)

DRAM是某些FPGA终端的外部可用存储器形式。DRAM提供大量的存储空间。但由于DRAM位于FPGA外部，应用不能在单个时钟周期内从DRAM获取数据。DRAM还需要连续的访问，意味着每次仅能使用一个命令访问内存。连续访问不能使用确定性时序，且根据等待访问DRAM的命令数量，可能增加应用的执行时间。

注： 不能执行使用DRAM的VI定义的存储器项。

DRAM可用时，用其存储无法存储在FPGA上的大型数据。如DRAM不可用于终端，**存储器属性**对话框的**实现**下拉菜单中不会列出DRAM选项。

相关概念：

- 实现多个时钟域
- 在子VI中使用I/O、时钟、寄存器项、存储器项、FIFO和握手项
- 判定何时使用重入或非重入子VI

创建FPGA存储器项

单时钟域内使用FPGA应用的主要数据存储方式为存储器项。通过**项目浏览器**窗口或程序框图创建存储器项。

通过项目浏览器窗口创建存储器项

按照下列步骤在**项目浏览器**窗口创建终端范围的存储器项：

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 选择**新建**»**存储器**，打开**存储器属性**对话框。
3. 在**常规**页面中，展开**实现**下拉菜单以显示可用的存储器选项。

4. 单击**确定**按钮完成创建存储器项。
5. 从**项目浏览器**窗口拖曳存储器项至程序框图。LabVIEW添加为存储器项配置的存储器方法节点至程序框图。

通过程序框图创建存储器项

通过程序框图可创建终端范围或VI定义的存储器项。

终端范围

1. 显示FPGA VI的程序框图。
2. 在**函数**选板中，添加**存储器方法节点**至程序框图。
3. 右键单击存储器方法节点，选择**添加新存储器**，显示**存储器属性**对话框。

> **提示** 右键单击存储器方法节点，选择**选择存储器»x**。其中**x**为已有的存储器项。

4. 在**常规**页面中，展开**实现**下拉菜单以显示可用的存储器选项。
5. 单击**确定**按钮完成创建存储器项。

或者连线**存储器输入**的引用，指定已有的存储器项。指定存储器后，右键单击“存储器方法节点”，选择**选择方法»y**。其中**y**为指定的方法。

通过VI定义

1. 显示FPGA VI的程序框图。
2. 在**函数**选板中，添加**通过VI定义存储器配置节点**至程序框图。
3. 右键单击“通过VI定义存储器配置”节点，选择**配置**显示**存储器属性**对话框。
4. 展开**实现**下拉菜单显示可用的存储器选项。
5. 单击**确定**按钮完成创建存储器项。

在程序框图上配置节点，以使用通过VI定义的存储器项时，LabVIEW将追加VI::至存储器项的名称。例如，命名通过VI定义的存储器项为Coefficients，LabVIEW将显示该存储器项的名称为VI::Coefficients。

使用DRAM

某些FPGA终端包含板载动态随机访问内存(DRAM)，用户可通过FPGA VI直接访问这些内存。LabVIEW支持两种类型的DRAM接口：

- **FPGA存储器项**－在项目中使用存储器项接口，此时使用DRAM的方法与存储器块和LUT的使用方法类似。DRAM存储器项位于项目浏览器窗口的FPGA终端下。
- **插槽CLIP**－在项目中使用CLIP接口，以直接与板载DRAM通信。插槽CLIP列出了所有与选中DARM存储器项兼容的内存接口。

多数使用DRAM的FPGA应用可从FPGA存储器项接口提供的可用性和VHDL优化特性中受益。但仍可使用插槽CLIP接口访问使用I/O信号的原地址。不能同时使用FPGA存储器项和插槽CLIP访问VI的同一个DRAM内存。不能使用通过VI定义的存储器项配置DRAM。

判定DRAM是否可用

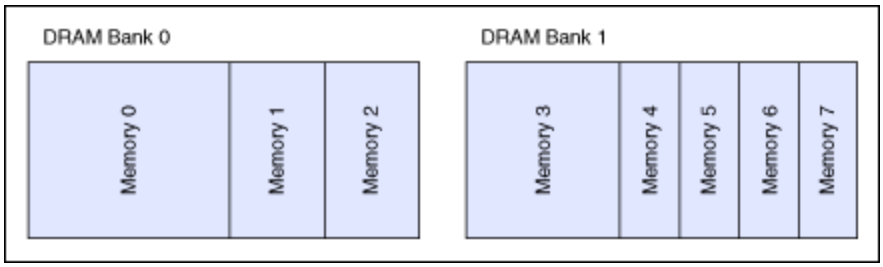
按照下列步骤判定DRAM在用户终端上是否可用。

1. 显示**项目浏览器**窗口。
2. 右键单击FPGA终端，选择**属性**。
3. 如用户终端的DRAM可用，**DRAM属性**将显示在对话框左侧的**类别**列表中。

注： 如DRAM不可用于终端，**存储器属性**对话框的**实现**下拉菜单中不会列出DRAM选项。

划分物理DRAM为多个存储器项

FPGA存储器项接口可用于划分终端上可用的物理DRAM内存为多个存储器项。使用存储器属性对话框创建和配置DRAM块上的内存分区。例如，如终端带有2个物理DRAM内存条，用户可将其中一条内存划分为3块存储器，另一条内存划分为5块存储器。如下图所示。LabVIEW将每块存储器视作独立的部分。



创建存储器分区后，必须使用“FPGA终端属性”对话框的DRAM属性页面配置同一DRAM内存条中的分区仲裁。

对DRAM内存条的仲裁访问

划分DRAM为多个存储器项后，可配置LabVIEW访问每个分区的时间。默认情况下，LabVIEW为所有分区分配相等的时间。DRAM判定器与同一共享资源的判定不同请求的判定器不是同一个判定器。

在FPGA终端属性对话框的DRAM属性页面中指定分配给每个分区的时间。LabVIEW使用轮叫调度分配用于每个分区的时间。

延迟和DRAM

DRAM的访问包含一些非确定性的延迟。使用请求数据和获取数据方法从DRAM读取数据可补偿该数据延迟。使用请求数据方法可排序多个访问数据的请求，通过获取数据方法可获取请求的数据。通过握手信号示意可接收数据时，DRAM将返回请求的数据。

如要优化DRAM性能，请以脉冲形式发送数据请求或写入数据至DRAM。

相关概念：

- 管理共享的资源
- 使用握手信号控制定时

通过双端口读取降低存储器资源使用

对于某些应用，可通过在“存储器属性”对话框的“接口”页面配置用于双端口读取访

问的存储器，减少块存储器资源的使用量和/或缩短执行时间。

双端口读取存储器的主要优点在于通过两个读取端口，用户可同时访问两个不同地址的数据。例如，同一正弦或余弦函数的系数。用户可将该函数在存储器块中保存一次，然后通过两个地址读取不同的相位值。两个读取该存储器的存储器方法节点可位于同一结构中（例如，单周期定时循环内）或FPGA VI的不同位置。

相关概念：

- 优化FPGA VI的执行速度和大小
- 在FPGA终端上存储数据

存储和访问FPGA设计不同部分的数据

当需要通过多个时钟域访问通过FPGA应用存储的数据时，请选择下列选项：

方法	使用场景
寄存器项	• 需要存储单个数据点 • 数据传输可能发生损耗 • 每个时钟周期需要发布状态信息
使用存储器块实现存储器项	• 需要存储多个数据点 • 数据传输可能发生损耗
握手项	• 读取方和写入方节点同步 • 数据传输不能发生损耗

寄存器项

在下列情况下使用寄存器项存储数据：需要跨越多个时钟域访问数据、需要访问设计不同部分的数据以及需要写入可重入代码。相对于FIFO，寄存器项占用较少的

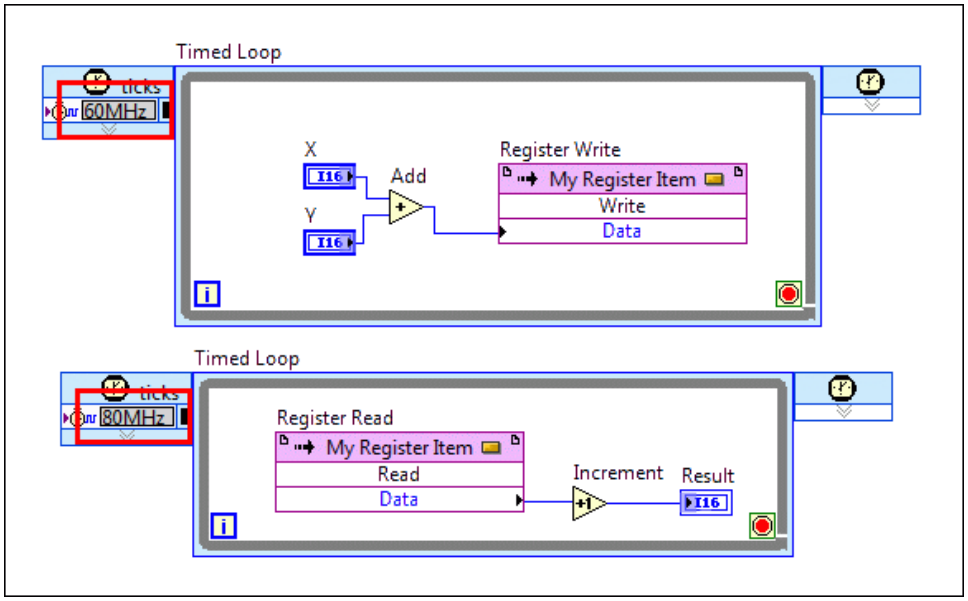
FPGA资源。并且寄存器项不会占用稀缺的FPGA资源（存储器块）。

> **注：** 寄存器项不支持自定义类型。如应用需要使用自定义类型，请使用全局变量替代寄存器项。

LabVIEW FPGA模块具有两种寄存器项类型：

- **通过VI定义的寄存器项**：使用VI定义的寄存器项创建重入子VI并避免资源冲突。在重入FPGA VI中配置通过VI定义的寄存器项，LabVIEW将为用于VI的每个实例的寄存器项创建独立的副本。
- **终端范围的寄存器项**：如需寄存器项可见且可通过**项目浏览器**窗口配置，请使用终端范围的寄存器项。终端范围的寄存器项在**项目浏览器**窗口的同一终端下的任意FPGA VI内均可用。

下列程序框图为使用寄存器方法节点读取和写入寄存器项的方法，该节点配置用于终端范围的寄存器项。此VI可将**X**和**Y**两个控件的值相加，将和值写入寄存器项，读取存储的值，将存储的值递增加1并将结果传递至显示控件。注意，“写入”方法和“读取”方法位于两个不同的时钟域内。



通过项目浏览器窗口创建终端范围的寄存器项

按照下列步骤通过**项目浏览器**窗口创建终端范围的寄存器项。

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 选择**新建»寄存器**，显示**FPGA寄存器属性**对话框。
3. 单击**确定**，完成创建寄存器项。
4. 从**项目浏览器**窗口拖拽寄存器项至程序框图。LabVIEW添加一个用于寄存器项的寄存器方法节点。

注： 如使用该方法创建寄存器项，寄存器项按名称捆绑至项目中的相应项。即更新项目的项将影响程序框图上的全部寄存器项实例。

通过程序框图创建寄存器项

通过程序框图可创建终端范围的或VI定义的寄存器项。

终端范围：

1. 显示FPGA VI的程序框图。
2. 在**函数**选板中，添加寄存器方法节点至程序框图。
3. 右键单击寄存器方法节点，选择**添加新寄存器**，显示**FPGA寄存器属性**对话框。

提示也可以右键单击寄存器方法节点，选择**选择寄存器»x**，其中x为已有的寄存器项。

4. 单击**确定**，完成创建寄存器项。

也可以连线**寄存器输入**输入端的引用，指定现有的寄存器项。LabVIEW将配置寄存器方法节点为默认的写入方法。指定寄存器项后，右键单击寄存器方法节点，选择**选择方法»y**，其中y为指定的方法。

通过VI定义：

1. 显示FPGA VI的程序框图。

2. 在**函数**选板中，添加通过VI定义寄存器配置节点至程序框图。
3. 右键单击通过VI定义寄存器配置节点，选择**配置**，显示**FPGA寄存器属性**对话框。
4. 单击**确定**，完成创建寄存器项。

握手项

握手项与单元素FIFO操作类似。使用握手项在写入方节点和读取方节点间传输单个数据元素，写入方节点与读取方节点可位于同一时钟域或不同时钟域。使用握手项在写入域和读取域间实现无损耗的传输，并在读取方接收到数据时通知写入域。相对于FIFO，握手项占用较少的FPGA资源。并且握手项不会占用稀缺的FPGA资源（存储器块）。

LabVIEW FPGA模块具有两种握手项类型：

- **通过VI定义的握手项**：使用通过VI定义的握手项创建重入子VI并避免资源冲突。在重入FPGA VI中配置通过VI定义的握手项，LabVIEW将为用于VI的每个实例的握手项创建独立的副本。
- **终端范围的握手项**：如需握手项可见且可通过**项目浏览器**窗口配置，请使用终端范围的握手项。终端范围的握手项在**项目浏览器**窗口中同一终端下的任意FPGA VI内均可用。

使用握手方法节点读取数据

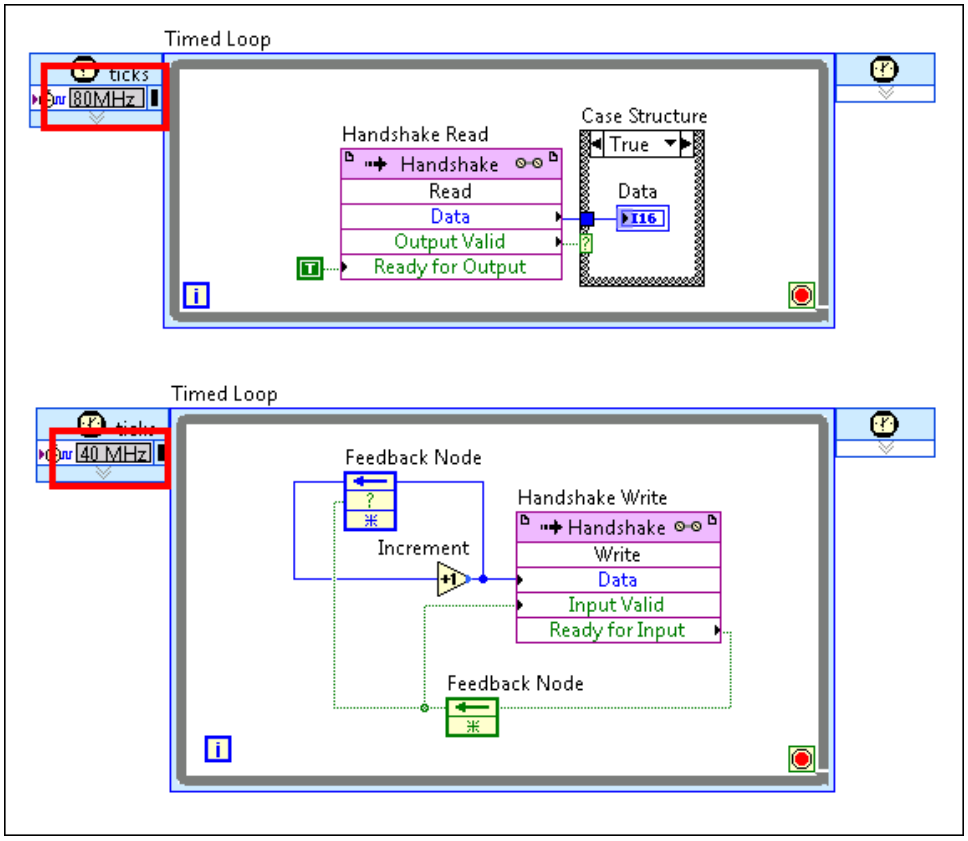
使用握手方法节点有两种读取数据的方法：

方法	使用场景
读取方法	• 仅需读取数据元素一次。**读取**方法会自动确认数据，即它将移除数据，为下一次传输做好准备。
确认的读取方法和无需确认的读取方法	• 节点确认数据传输时需要执行控制。读取方节点可读取同一数

方法	使用场景
	据元素多次，直至确认失效。

下列程序框图为使用握手方法节点读取和写入握手项的方法，该节点配置用于终端范围的握手项。该VI将值每次递增1，将数据写入握手项，读取存储的值并自动通知读取方。**输入就绪**为TRUE时，**输入有效**为TRUE。当**输出有效**为TRUE时，VI会将存储的值写入显示控件。注意，“写入”方法和“读取”方法位于两个不同的时钟域内。如红框所示。该设计确保将全部数据点写入至更快的时钟域，且无数据损耗。

注： 中止并重新启动VI，不会重置握手项。使用**清除**方法将握手项返回为默认状态。



通过“项目浏览器”窗口创建终端范围的握手项

按照下列步骤通过**项目浏览器**窗口创建终端范围的握手项：

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 选择**新建»握手**，显示**FPGA握手属性**对话框。使用对话框中的选项配置握手项。
3. 单击**确定**，完成创建握手项。
4. 从**项目浏览器**窗口拖拽握手项至程序框图。LabVIEW添加一个用于握手项的握手方法节点。

注： 如使用该方法创建握手项，寄存器项按名称捆绑至项目中的相应项。即更新项目的项将影响程序框图上的全部握手项实例。

通过程序框图创建握手项

通过程序框图可创建终端范围的或通过VI定义的握手项。

终端范围：

1. 显示FPGA VI的程序框图。
2. 添加握手方法节点至程序框图。
3. 右键单击握手方法节点，选择**添加新握手**，显示**FPGA握手属性**对话框。

提示也可以右键单击握手方法节点，选择**选择握手»x**，其中**x**为已有的握手项。

4. 单击**确定**，完成创建握手项。

也可以连线**握手输入**输入端的引用，指定握手项。LabVIEW会将握手方法节点配置为默认的模式（写入）。指定握手项后，右键单击握手方法节点，选择**选择方法»y**，其中**y**为指定的方法。

通过VI定义：

1. 显示FPGA VI的程序框图。
2. 添加握手方法节点至程序框图。
3. 右键单击通过VI定义握手配置节点，显示**FPGA握手属性**对话框。
4. 单击**确定**，完成创建握手项。

相关概念：

- 实现多个时钟域
- 在子VI中使用I/O、时钟、寄存器项、存储器项、FIFO和握手项
- 判定何时使用重入或非重入子VI

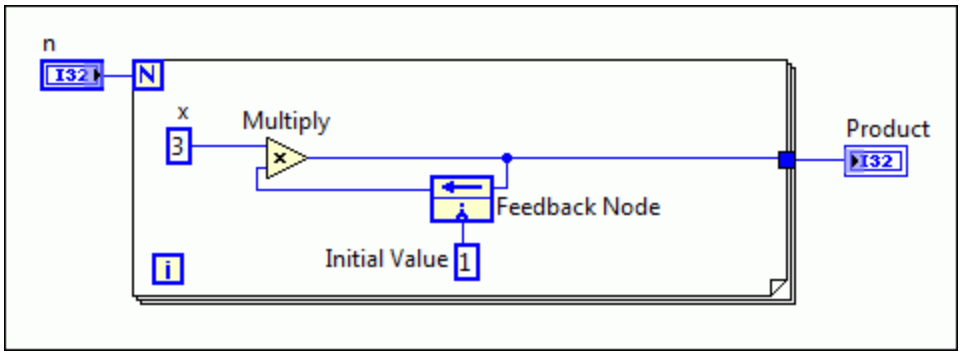
在循环间存储数据

反馈节点

反馈节点使用FPGA资源存储数据。使用反馈节点存储一个VI执行或循环的数据（例如，状态信息）至下一个执行或循环。反馈节点接收到新值时，节点将保留该值直到节点将该值传递至下个输出接线端。如要存储多个连续计数的数据采样，在属性对话框的配置页面增加**延时**的值，以延迟反馈节点的输出。

> **注：** 离散延时函数与反馈节点类型。在某些情况下更适合使用离散延时函数。关于判定适用于用户的函数的帮助主题，见离散延时函数帮助主题中的表格。

下列范例VI使用“反馈”节点。VI运行For循环，循环计数值为**n**。VI将反馈节点中存储的值乘以常量值`3`，然后返回结果。初次调用VI时，“反馈”节点的值为`1`。假设计数值为n，返回的**结果**为3^n。



按照下列步骤创建反馈节点。

1. 显示FPGA VI的程序框图。
2. 在函数选板中，选择**编程**»**架构**»**反馈节点**并添加至程序框图。

连线子VI、函数、一组子VI或函数至同一VI、函数或组时将自动出现“反馈”节点。

通过**属性**对话框配置“反馈”节点。右键单击节点并从快捷菜单中选择属性可显示该对话框。如在该对话框的配置页面勾选了**忽略FPGA重置方法**复选框，LabVIEW FPGA 将在底层寄存器实例中移除重置。移除后，编译器可使用移位寄存器查找表(SRL)，而不是触发器实现延时。SRL可在单个查找表(LUT)中包含多个延迟，从而节省了大量的FPGA资源。

在循环中，右键单击“反馈”节点，在快捷菜单中选择**替换为移位寄存器**将反馈节点替换为移位寄存器。反之，移位寄存器也同样可替换为反馈节点。但不能实现使用SRL的移位寄存器。

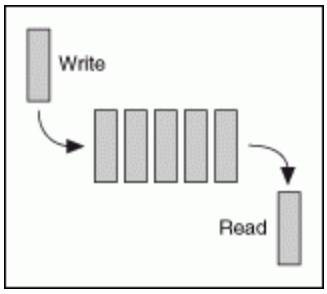
相关概念：

- 在FPGA VI中交互AXI IP

在设备或FIFO架构间使用FIFO传输数据

FIFO可用于在FPGA VI的不同代码间、FPGA终端上的VI间及设备间传输数据。FIFO是一种以其接收到元素的顺序保存元素，并提供先入先出访问机制的数据架构。

下列示意图演示了通过FIFO的元素的动作。



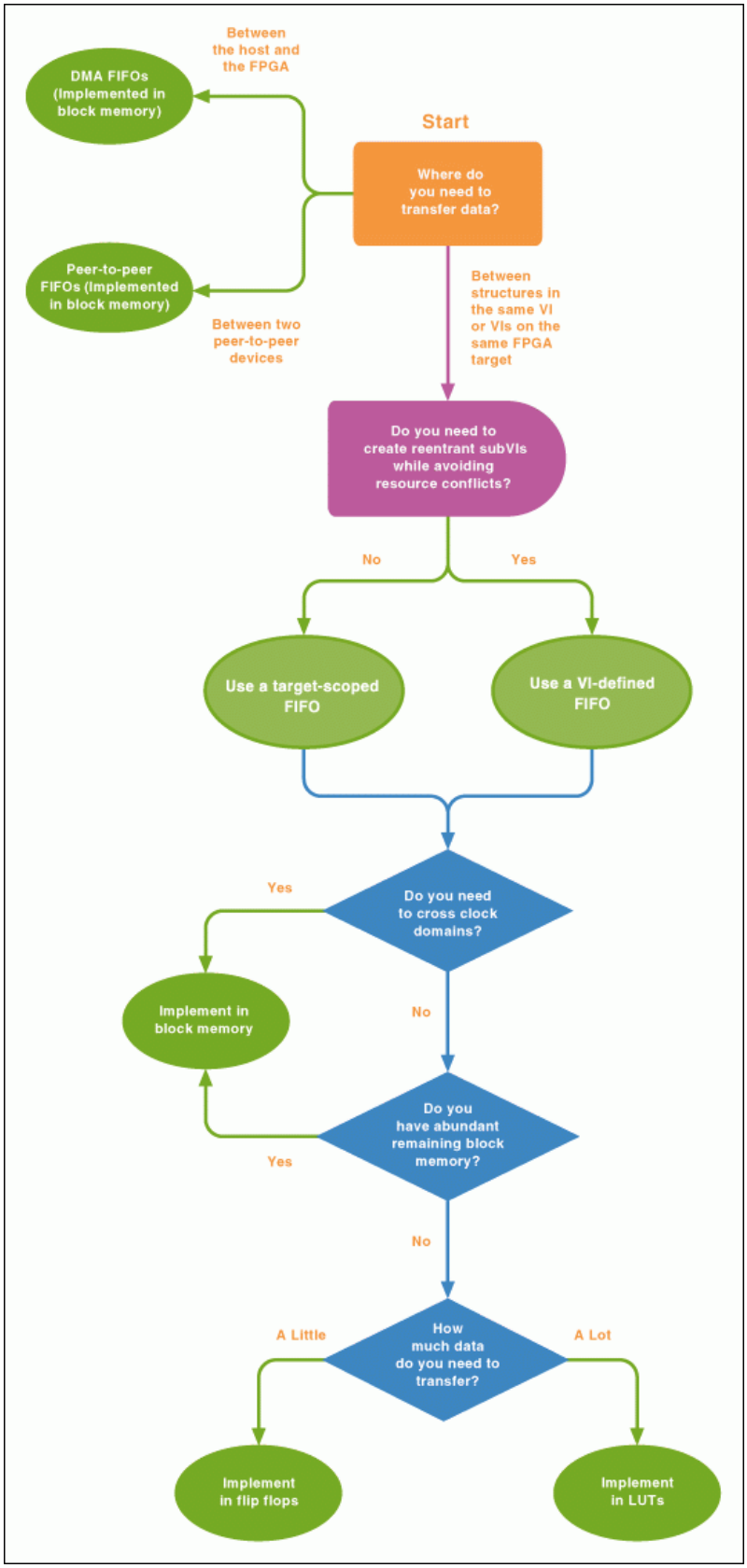
通过下列方式使用FIFO传输数据：

- 在一个时钟域的并行循环间
- 时钟域间
- 点对点终端间
- 主控计算机和FPGA间

通过FIFO属性对话框创建和配置FIFO。

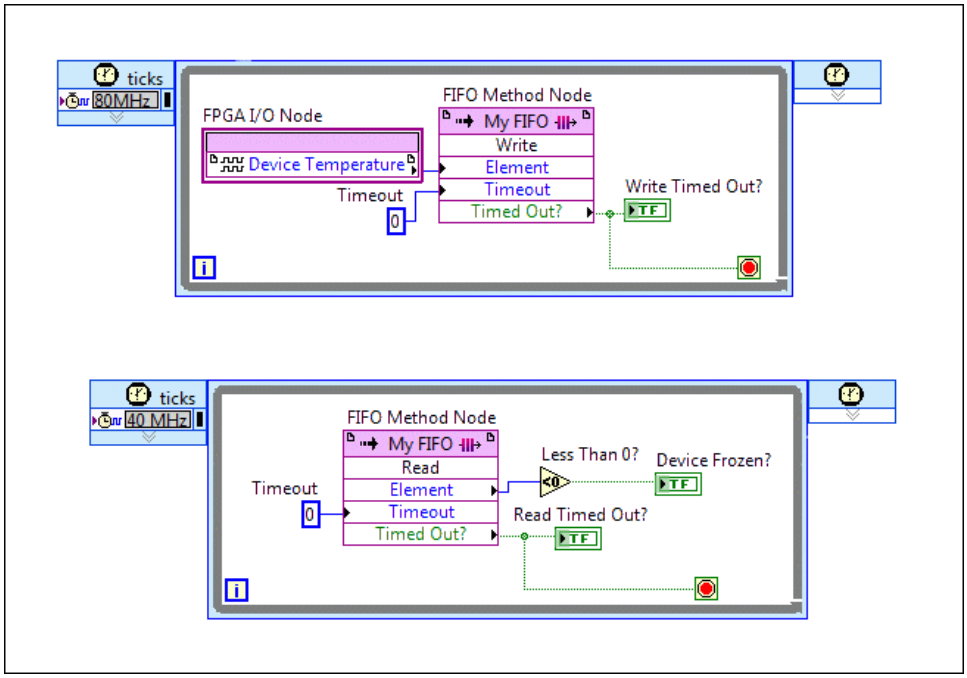
除通过VI定义的FIFO外，全部FIFO在项目中均具有相应的项。因此如使用通过VI定义FIFO外的FIFO，并将FPGA VI发送给另一用户时，必须发送整个项目。否则，该FPGA VI不能运行。

通过下列流程图判定最适合应用的FIFO配置。


Start
Where do you need to transfer data?
Between the host and the FPGA
DMA FIFOs (Implemented in block memory)
Peer-to-peer FIFOs (Implemented in block memory)
Between two peer-to-peer devices
Between structures in the same VI or VIs on the same FPGA target
Do you need to create reentrant subVIs while avoiding resource conflicts?
No
Yes
Use a target-scoped FIFO
Use a VI-defined FIFO
Do you need to cross clock domains?
Yes
Implement in block memory
No
Do you have abundant remaining block memory?
Yes
No
How much data do you need to transfer?
A Little
A Lot
Implement in flip flops
Implement in LUTs

注： 通过FPGA终端属性对话框的常规页面的**终端信息**组件，获取FPGA终端支持的内存块的数量信息。

下图为用于在同一个FPGA VI的两个单周期定时循环间传输数据的终端范围的FIFO。单周期定时循环位于两个不同的时钟域内。顶层循环使用80 MHz时钟，底层循环使用40 MHz时钟。在该范例中， 位于80 MHz时钟域内的FPGA I/O节点通过设备获取温度读数。FIFO方法节点写入数据至FIFO。如FIFO中的空元素不能立即写入数据，节点超时。应中止循环并设置**写入超时?** 为TRUE。位于40 MHz时钟域的FIFO方法节点读取该数据，然后传输数据至“小于零？”函数。如温度小于零，函数设置**设备结冰?** 显示控件为TRUE。如不能立即读取数据，FIFO方法节点超时。循环将终止并设置**读取超时?** 显示控件的值为TRUE。



注： 如FIFO方法节点包含一个**超时**输入参数，节点位于单周期定时循环内时必须连线参数至一个零常量。

数据传输任务

通过下列方式传输数据：

- **在单个FPGA终端间**—使用上述FIFO类型在VI间传输数据、传输数据至单个VI的循环或从单个VI传输数据至循环或在时钟域间传输数据。
  - **终端范围的FIFO**—如要FIFO在**项目浏览器**窗口可见并可配置，使用终端范围的FIFO。终端范围的FIFO在项目中具有相应的项。更新项目项将影响FIFO的全部实例。在**项目浏览器**窗口中，同一FPGA终端下的任意VI内的终端范围的FIFO均可用。如使用终端范围的FIFO并想要将其发送给另一用户，必须发送整个项目。否则FPGA VI将显示为断开。
  - **通过VI定义FIFO**—使用通过VI定义FIFO和FIFO名称控件，创建可重入子VI及避免资源冲突。在可重入FPGA VI中配置VI定义的FIFO时，LabVIEW为每个VI的实例创建独立的FIFO的副本。
- **在主机和FPGA间**—使用直接内存访问(DMA) FIFO在主机和FPGA间传输大型的数据。该FIFO类型直接访问内存以从FPGA终端VI传输数据至主控VI，反之亦然。DMA FIFO在主机和FPGA终端上均分配内存，但仍用作单个FIFO。DMA FIFO通过在主机和FPGA间使用前面板输入控件和显示控件，提供了性能优势。
- **在两个点对点终端间**—使用点对点FIFO在点对点终端间传输数据，无需通过主机发送数据。点对点数据流可用于在FPGA终端和非FPGA终端间传输数据，但终端必需支持使用点对点数据流架构。点对点的FIFO在项目中具有相应的项。

注： 如包含多个至同一FIFO的读取或写入，FIFO可作为共享资源。如要防止数据丢失和抖动，应避免同时发送读取或写入请求至单个FIFO。

相关概念：

- 存储和传输数据
- 使用直接内存访问传输数据

在FPGA VI中创建FIFO

LabVIEW提供不同类型的FIFO，用于在FPGA VI的不同部分间、FPGA终端的VI间或设备间传输数据。可使用**项目浏览器**窗口创建FIFO，或在程序框图中直接创建终端范围和通过VI定义的FIFO。

通过项目浏览器窗口创建终端范围的FIFO

按照下列步骤在**项目浏览器**窗口中创建终端范围的FIFO。

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 选择**新建»FIFO**，显示“FIFO属性”对话框。
3. 在“常规”页面中，展开**实现**下拉菜单以显示可用的选项。
4. 单击**确定**按钮完成创建FIFO。
5. 从**项目浏览器**窗口拖曳FIFO至程序框图。LabVIEW添加为FIFO配置的FIFO方法节点至程序框图。

通过程序框图创建FIFO

通过程序框图可创建终端范围或VI定义的FIFO。

终端范围：

1. 显示程序框图。
2. 在“函数”选板中，添加FIFO方法节点至程序框图。
3. 右键单击FIFO方法节点，从快捷菜单中选择**添加新FIFO**，显示“FIFO属性”对话框。

> **提示** 也可以右键单击FIFO方法节点，从快捷菜单中选择**选择FIFO»x**，其中**x**为现有的FIFO。

4. 在**常规**页面中，展开**实现**下拉菜单以显示可用的选项。
5. 单击**确定**按钮完成创建FIFO。

也可连线FIFO常量、FIFO方法节点或通过VI定义FIFO配置节点至**FIFO输入**的输入端，以指定FIFO。LabVIEW将FIFO方法节点配置为默认的方法。对于支持写入方法的FIFO，默认方法为写入。如要在指定FIFO后选择不同的方法，可右键单击FIFO方法节点，在快捷菜单中选择**选择方法»y**，其中**y**为指定的方法。

通过VI定义：

1. 显示程序框图。
2. 在“函数”选板中，添加通过VI定义FIFO配置节点至程序框图。
3. 右键单击通过VI定义FIFO配置节点，从快捷菜单中选择**配置**，显示“FIFO属性”对话框。
4. 在“常规”页面中，展开**实现**下拉菜单以显示可用的选项。
5. 单击**确定**按钮完成创建FIFO。

通过项目浏览器窗口创建DMA FIFO

首先，按照下列步骤判定终端是否支持DMA FIFO。

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 从快捷菜单中选择**属性**，显示“FPGA终端属性”对话框。
3. 查看“常规”页面的**终端信息**框，查找DMA支持信息。若终端支持DMA，**终端信息**框将显示DMA通道的数量。若终端不支持DMA，**终端信息**框将显示不支持DMA。

如终端支持DMA FIFO，按照下列步骤在FPGA VI中创建DMA FIFO。关于配置主控VI读取和写入DMA FIFO的帮助信息，见从主控VI读取DMA FIFO和从主控VI写入DMA FIFO。

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 选择**新建»FIFO**，显示“FIFO属性”对话框。
3. 根据流式传输数据的方向，在**类型**下拉菜单中选择**主机至终端 – DMA**或**终端至主机 – DMA**。
4. 单击**确定**按钮完成创建FIFO。
5. 从**项目浏览器**窗口拖曳FIFO至程序框图。LabVIEW添加为FIFO配置的FIFO方法节点至程序框图。

通过项目浏览器窗口创建点对点FIFO

按照下列步骤通过**项目浏览器**窗口创建点对点FIFO，或判定特定终端是否支持点对点数据流。

1. 在**项目浏览器**窗口，右键单击FPGA终端。
2. 选择**新建»FIFO**，显示“FIFO属性”对话框。
3. 在“常规”页面中，从**类型**下拉菜单中选择**点对点写入方**或**点对点读取方**。如下拉菜单中不包含上述选项，即终端不支持点对点FIFO。
4. 单击**确定**按钮完成创建FIFO。

关于如何使用点对点FIFO的信息，见在FPGA终端上使用点对点数据流主题。

选择FIFO实现选项

理解可用的FIFO实现选项能够帮助您创建更高效的FPGA设计。FIFO实现选项指定FPGA在硬件中表示终端范围和通过VI定义FIFO的方法。可通过FIFO属性对话框选择FIFO实现选项。

注： DMA和P2P FIFO始终使用块存储器。

下表能够帮助您判定适合自身应用的FIFO实现方式。

实现	实现内容	注意事项
触发器	触发器触发器是可用于存储数据或执行其他任务（例如，加法和减法）的FPGA资源。因此触发器通常为移动或存储数据的紧缺FPGA资源。	不能使用通过跨多个时钟域的触发器实现的FIFO。
LUT（查找表）	LUT（即分布式RAM）由FPGA上硬连线的逻辑门组成。与触发器一样，LUT也可以作为FPGA的资源。	不能使用通过跨多个时钟域的查找表实现的FIFO。
块存储器	块存储器（也称为块RAM、BRAM）是专门用于数据存储的FPGA资源。关于配合使用块存储器和内置控制逻辑的详细信息，请参考实现块存储器FIFO。使用存储器块配置终端范围或VI定义的FIFO时，可配置控	选择**块存储器**选项时，写入数据至FIFO后至少要经过6个时钟周期，才能读取在终端范围或通过VI定义的FIFO中读取数据。选择超时接口时，连接FIFO方法节点（配置为读取方法）的**超时?**输出，以判定上游数据是否已就绪。选择握手接口时，连接FIFO方法节点（配置为读

	制逻辑。DMA和P2P FIFO只能使用存储器块。	取方法）的**输出有效**输出，以判定上游数据是否已就绪。
UltraRAM	UltraRAM (URAM)是一种高度灵活的低密度大容量存储块，可用于大多数Xilinx UltraScale+终端。UltraRAM的容量是块存储器的8倍，但数据宽度和地址空间配置的灵活性不如块存储器。UltraRAM可用于存储较大的本地FIFO。	不能跨多个时钟域使用实现方式为UltraRAM的FIFO。“获取读取元素数量”、“获取写入元素数量”方法不支持实现方式为UltraRAM的FIFO。

相关概念：

- 实现多个时钟域
- 实现存储器块FIFO

选择FIFO接口选项

理解可用的FIFO接口选项，可减少实现FIFO超时和握手协议所需的代码。FIFO接口选项更改指定FIFO方法的可用输入和输出。要选择超时或握手接口，可在单周期定时循环内，右键单击调用读取或写入方法的FIFO方法节点，然后从快捷菜单中选择**接口»超时**或**接口»握手**。

下表能够帮助您判定适合自身应用的FIFO接口。

接口	实现内容	使用场景
超时	指定方法在返回超时前，等待操作完成的时间。该时间以时钟滴答为单位。超时接口通过启用输入和输出端（**超时**和**超时?**），帮助用户处理不响应的设备或代码段。	该接口为全部调用读取、写入或刷新并禁用方法的FIFO的默认接口。
握手	在高吞吐率应用中方便上方节点和下方节点间的通信。握手接口通过启用FIFO方法节点的下列输入和输出，减少了实现握手协议所需的逻辑量：**输入有效**、**输出就绪**、**输出有**	握手接口仅适用于单周期定时循环内的调用读取或写入方法的FIFO。此外，某些终端不支持用于点对点和DMA FIFO的握手接口。对于不支持握手接口的FIFO，LabVIEW在编译

	效和**输入就绪**。	时返回错误。

清除FPGA FIFO

在带实际I/O的开发计算机上运行FPGA VI时，若FPGA VI停止后又重新开始，LabVIEW将清除FIFO。在FPGA终端上使用交互式前面板通信运行FPGA VI时，若FPGA VI停止后又重新开始，LabVIEW不会清除FIFO。如要在FPGA上清除终端范围或VI定义的FIFO，可使用FIFO方法节点中的清除方法。也可以在**项目浏览器**窗口中右键单击VI，从快捷菜单中选择**下载**以清除FIFO。

使用可编程式FPGA接口通信控制FPGA VI时，可使用调用方法函数的“停止”方法从主控VI中清除每个DMA FIFO。也可以使用调用方法函数的重置方法，或勾选了**在最后引用时关闭或重置**快捷菜单选项的关闭FPGA VI引用函数，清除所有FIFO。除清除FIFO外，重置方法和**在最后引用时关闭或重置**选项还可完成其它任务。

相关概念：

- 使用仿真模式调试FPGA VI

实现存储器块FIFO

实现存储器块FIFO时，FIFO可容量的实际元素数量取决于下文提及的因素。

通过逻辑架构控制逻辑实现FIFO

如使用通过**逻辑片架构**控制逻辑实现的存储器块实现FIFO，下表中因素将影响FIFO可容纳的实际元素数量：

FIFO类型	待指定大小	考虑因素
终端	2的幂加少量元素	常规页显示2^M+5，M是地址宽度。LabVIEW强制转换**请求的元素数量**为最接近2^M+5的值。例如，**请求的元素数量**为1,000，LabVIEW强制转换该值

FIFO类型	待指定大小	考虑因素
范围的FIFO		为1,029。如FPGA没有足够的资源来强制转换**请求的单元数量**，FPGA VI将编译失败。
DMA FIFO	• **终端至主机** - DMA的大小为2的幂减1 • **主机至终端** - DMA的大小为2的幂加上6倍数待读取元素数量值减1	**常规**页显示2^M-1或2^M+（6 × 待读取元素数量）-1，M是地址宽度。LabVIEW将**请求元素数量**强制向上转换为最近有效值。如FPGA没有足够的资源来强制转换**请求的单元数量**，FPGA VI将编译失败。必须使用通过**逻辑片架构**控制逻辑实现的存储器块来实现DMA FIFO。最大DMA FIFO容量随终端变化。关于DMA FIFO大小限制的详细信息，见指定FPGA终端的硬件文档。
点对点FIFO	• 写入方FIFO的大小为2的幂减1 • 读取方FIFO的大小为2的幂加上6倍数待读取元素数量值减1	**常规**页显示2^M-1或2^M+（6 × 待读取元素数量）-1，M是地址宽度。LabVIEW将**请求元素数量**强制向上转换为最近有效值。如FPGA没有足够的资源来强制转换**请求的单元数量**，FPGA VI将编译失败。必须使用通过**逻辑片架构**控制逻辑实现的存储器块来实现点对点FIFO。

通过内置控制逻辑实现FIFO

如使用通过**内置**控制逻辑实现的存储器块实现FIFO，FIFO可容量的实际元素数量受到下列限制：

- 启用写入方法的握手接口将增加一个FIFO深度。
- **终端范围的FIFO** - LabVIEW强制转换**请求的元素数量**为可使用内置FIFO实现的深度。强制转换的**请求的元素数量**的计算随FPGA设备系列变化。如FPGA没有足够的资源来强制转换**请求的单元数量**，FPGA VI将编译失败。

注： 仿真输出不支持内置的FIFO和终端优化的FIFO。用户可使用条件禁用结构实现用于仿真输出的逻辑片架构FIFO。

实现带终端优化控制逻辑的FIFO

如使用通过**终端优化**控制逻辑实现的存储器块实现FIFO，本主题提及的全部限制均会影响FIFO可容量的实际元素数量。因为终端优化控制逻辑为内置逻辑和逻辑片架构控制逻辑的组合。

相关概念：

- 选择FIFO实现选项

使用内置的控制逻辑实现FIFO

FPGA硬件终端上的块内存可部署内置FIFO控制逻辑，从而节省大量的FPGA资源。另外，这些专用的电路可支持比使用片式结构创建的控制逻辑更高的时钟速率。

注： 关于终端上FPGA属性的详细信息，请参考Xilinx文档 。

使用内置FIFO控制逻辑的注意事项

以下列表概述了使用内置FIFO的注意事项。

- FPGA的每个块内存必须有一个专门的FIFO控制器。
- 内置FIFO的时钟域必须有稳定的和自由运行的时钟。
- (Xilinx Vivado) 下列方法不支持具有内置控制逻辑的FIFO：
  - 获取待读取元素数量
  - 获取待写入元素数量
- 使用下列方法会增加读写操作的资源使用和降低时钟域的频率：
  - 获取待读取元素数量
  - 获取待写入元素数量
  - 清除
- “清除”方法将使用多个时钟周期。因为它要依次清除元素而非同步清除元素。
- 在部分终端上，仿真导出不支持内置和终端优化的FIFO控制逻辑。用户可使用条件禁用结构实现用于仿真输出的片式FIFO控制逻辑。
- 启用写入方法的握手接口将增加一个FIFO深度。
- 当内置FIFO完全充满后，必须等到FIFO中至少有两个空白元素才能继续写入FIFO。“获取要写入元素数量”和“写入（FIFO方法）”指示FIFO已满，直到至少有两个空元素。
- 强制转换的**请求的元素数量**的计算随FPGA设备系列变化。

利用目标优化选项优化FIFO控制逻辑

如要使用内置FIFO，但是不保证应用程序满足使用内置FIFO的所有条件，可配置FIFO使用最优配置。在FIFO属性对话框的常规页面上，从**控制逻辑**下拉菜单中选择**终端优化**。

终端优化选项根据终端和应用程序的功能，选择**逻辑片架构**或**内置**选项。当终端支持内置FIFO，且重置时时钟域不停止时，应用程序使用内置FIFO。否则，应用程序使用逻辑片结构来创建控制逻辑。

> **注：** 选择**终端优化**表示FIFO中元素的实际数量根据应用中FIFO的目标和位置而变化。**实际元素数量**的值始终大于或等于**请求的元素数量**的值。

相关概念：

- 使用第三方仿真器调试FPGA VI

使用NI扫描引擎和变量传输数据

通过可编程式前面板通信传输相干FPGA I/O数据集至RT主控VI时，必须创建同步FPGA VI与主控VI的程序框图代码。如FPGA终端支持NI扫描引擎，可使用NI扫描引擎同步数据传输。然后通过用户定义的I/O变量在FPGA VI和RT主控VI间传输数据。

注： I/O变量是一种共享变量，它使用NI扫描引擎对I/O数据进行单点访问。关于NI扫描引擎支持的详细信息，见指定FPGA终端的硬件文档。

使用NI扫描引擎减少用于访问和在FPGA I/O通道和RT主控VI间传输相干数据集的代码量。使用用户自定义I/O变量，发送数据至RT主控VI前和发送数据至FPGA VI后可在FPGA终端上处理数据。例如，创建一个执行下列步骤的应用：

1. 获取模拟I/O数据并对FPGA VI中的数据执行FFT
2. 传输处理后的数据至RT VI中的控制循环
3. 从RT控制循环传输输出的数据至FPGA，以用作物理I/O通道的输出

步骤2和3包含在FPGA VI和主控VI间传输数据的用户定义的I/O变量。

(CompactRIO)关于使用用户定义I/O变量的范例，见`labview\examples\CompactRIO\NI Scan Engine\Getting Started\User-Defined IO Variable - Basic\`目录下的`User-Defined IO Variable - Basic.lvproj`。

创建用户定义I/O变量

在**项目浏览器**窗口右键单击机箱项，从快捷菜单中选择**新建»用户定义变量**，新建一个I/O变量。但是，因为所有的I/O变量都不具有方向属性，必须将每个用户定义的I/O变量配置为**FPGA至主机**或**主机至FPGA**。

通过该方法创建的I/O变量将出现在标签为**用户定义变量**的容器中。

注： 每个机箱项仅能包含一个用户定义I/O变量的容器。但用户定义的I/O变量容器可包含多个用户定义的I/O变量。

用户定义I/O变量的说明

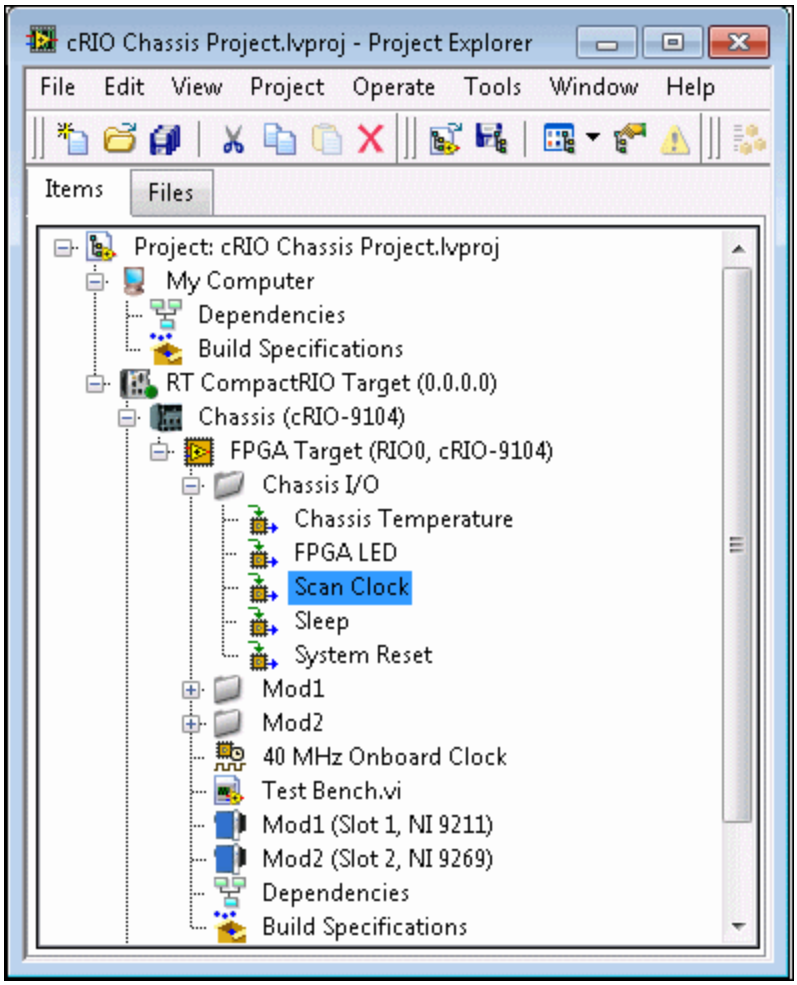
创建和使用用户定义I/O变量时请注意下列说明：

- 使用用户定义I/O变量前，必须打开FPGA VI的引用。
- 仅可在支持NI扫描引擎的FPGA终端上使用用户定义I/O变量。关于NI扫描引擎支持的详细信息见终端硬件文档。
- 用户定义I/O变量只支持扫描访问。不能直接访问用户定义I/O变量。
- 如添加用户定义I/O变量节点在FPGA VI程序框图上执行，必须设置FPGA VI的执行模式为FPGA终端。如要更改FPGA VI的执行模式，右键单击包含该VI的机箱项并选择**选择执行模式»FPGA终端**。如选择**仿真（仿真I/O）**或**仿真（实际I/O）**，且该FPGA VI包含用户定义I/O变量，**运行**按钮将显示断开且FPGA VI不能运行。
- 用户定义的I/O变量仅可用于运行在同一个机箱上的FPGA VI和RT VI间通信。如用户定义I/O变量已启用网络发布，可在任意RT VI或同一LabVIEW项目中的基于Windows的VI中使用该变量。例如，可使用网络发布I/O变量创建运行在Windows上的用户界面VI。

访问来自NI扫描引擎的定时信息

用户定义I/O变量基于来自NI扫描引擎的定时信息。通过添加Scan Clock I/O项至FPGA VI的程序框图可访问该定时信息。该I/O项传输来自扫描引擎的定时信息（例如，信号保持为高电平的FPGA时钟周期的数量）至FPGA VI。使用该定时信息设计应用程序，以确保在FPGA VI和RT主控VI间传输的数据集的一致性。

并非所有机箱均支持Scan Clock I/O项。如所用机箱不支持该项，它位于**项目浏览器**窗口**FPGA终端**的**机箱I/O**项下。下图为该项位置的示意图。



相关概念：

- 在FPGA和主机间传递数据

配合使用点对点数据流和FPGA终端

点对点数据流使用户能够直接从一个FPGA终端传输数据至另一个终端，而无需通过主机处理器发送数据。点对点数据流可用于在FPGA终端和非FPGA终端间传输数据，但终端必需支持使用点对点数据流架构。关于终端的点对点数据流功能的信息，见具体的FPGA终端文档。

按照下列步骤在两个终端间设置点对点数据流会话。

1. 创建点对点写入方FIFO

2. 创建点对点读取方FIFO
3. 创建点对点写入方FIFO的引用
4. 创建点对点读取方FIFO的引用
5. 创建点对点数据流会话

相关概念：

- 创建点对点的数据流会话
- 创建至点对点读取方FIFO的引用
- 创建点对点写入方FIFO的引用
- 创建点对点读取方FIFO
- 创建点对点写入方FIFO
- FPGA终端硬件文档

创建点对点写入方FIFO

按照下列步骤在FPGA终端上创建一个写入方FIFO，以用于点对点的数据流。对于非FPGA终端见指定FPGA终端文档，了解创建点对点数据流的详细信息。

注： 某些FPGA终端不支持点对点FIFO。关于终端是否支持点对点的详细信息，见FPGA终端属性对话框的常规页的**终端信息**对话框。

1. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**新建»FIFO**。
2. 在FIFO属性对话框中，从**类型**下拉菜单中选择**点对点写入方**。如下拉菜单中**点对点写入方**不可用，即终端不支持点对点FIFO。
3. 在**名称**文本框中输入名称。FIFO名称将出现在FIFO方法节点中。因此考虑命名FIFO使程序框图更具可读性。例如，可能需要使用名称`FIFO写入方`。
4. 完成配置FIFO后，单击**确定**按钮关闭对话框。
5. 为写入方创建一个FPGA VI。VI必须与点对点FIFO位于同一FPGA终端下。
6. 在主控数据流的VI中创建一个至写入方的引用。

相关概念：

- 创建点对点写入方FIFO的引用
- 配合使用点对点数据流和FPGA终端
- FPGA终端硬件文档

创建点对点读取方FIFO

按照下列步骤在FPGA终端上创建一个读取方FIFO，以用于点对点数据流。对于非FPGA终端见指定FPGA终端文档，了解创建点对点数据流的详细信息。

> **注：** 某些FPGA终端不支持点对点FIFO。关于终端是否支持点对点的详细信息，见FPGA终端属性对话框的常规页的**终端信息**对话框。

1. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**新建»FIFO**。
2. 在FIFO属性对话框中，从**类型**下拉菜单中选择**点对点读取方**。如下拉菜单中**点对点读取方**不可用，即终端不支持点对点FIFO。
3. 在**名称**文本框中输入名称。FIFO名称将出现在FIFO方法节点中。因此考虑命名FIFO使程序框图更具可读性。例如，可能需要使用名称`FIFO`读取方。
4. 完成配置FIFO后，单击**确定**按钮关闭对话框。
5. 为读取方创建一个FPGA VI。VI必须与点对点FIFO位于同一FPGA终端下。
6. 在主控数据流的VI中创建一个至读取方的引用。

相关概念：

- 创建至点对点读取方FIFO的引用
- 配合使用点对点数据流和FPGA终端
- FPGA终端硬件文档

创建点对点写入方FIFO的引用

创建点对点数据流会话前，必须创建一个点对点写入方FIFO的引用。如点对点写入方FIFO未位于FPGA终端上，请参阅具体的FPGA终端文档，了解创建引用的详细信息。对于FPGA终端，按照下列步骤创建点对点写入方FIFO的引用。

1. 为FPGA终端创建一个写入方FIFO。
2. 在主控VI的程序框图上，放置一个“打开FPGA VI引用”函数。
3. 从**项目浏览器**窗口将用于写入方的FPGA VI拖曳至“打开FPGA VI引用”函数。
4. 在程序框图上放置一个调用方法节点。
5. 连线“打开FPGA VI引用”函数的**FPGA VI引用输出**接线端至调用方法节点的**FPGA VI引用输入**接线端。
6. 单击“调用方法”节点的**方法**接线端，从快捷菜单中选择**FIFO»获取点对点写入方**。其中FIFO为FPGA终端下的点对点FIFO的名称。
7. 使用“调用方法”节点的**写入方**接线端的输出创建点对点数据流会话。

相关概念：

- 配合使用点对点数据流和FPGA终端
- FPGA终端硬件文档
- 创建点对点写入方FIFO
- 创建点对点的数据流会话

创建至点对点读取方FIFO的引用

创建点对点数据流会话前，必须创建一个至点对点读取方FIFO的引用。如点对点读取方FIFO未位于FPGA终端上，请参阅具体FPGA终端文档了解创建引用的信息。对于FPGA终端，按照下列步骤创建点对点读取方FIFO的引用。

1. 为FPGA终端创建一个读取方FIFO。
2. 在主控VI的程序框图上，放置一个“打开FPGA VI引用”函数。
3. 从**项目浏览器**窗口拖曳用于读取方的FPGA VI至打开FPGA VI引用函数。
4. 在程序框图上放置一个调用方法节点。
5. 连线“打开FPGA VI引用”函数的**FPGA VI引用输出**接线端至调用方法节点的**FPGA VI引用输入**接线端。
6. 单击调用方法节点的**方法**接线端，从快捷菜单中选择**FIFO»获取点对点读取方**，其中FIFO为FPGA终端下的点对点FIFO的名称。
7. 使用调用方法节点的**读取方**接线端的输出，从主控VI创建点对点数据流会话。

相关概念：

- 配合使用点对点数据流和FPGA终端
- FPGA终端硬件文档
- 创建点对点读取方FIFO
- 创建点对点的数据流会话

创建点对点的数据流会话

按照下列步骤创建一个点对点数据流会话。

1. 在程序框图上放置niP2P创建点对点数据流 VI。
2. 连线读取方的引用至niP2P创建点对点数据流VI的**读取方**输入接线端。
3. 连线写入方的引用至niP2P创建点对点数据流VI的**写入方**输入接线端。
4. 连线niP2P创建点对点数据流VI的**数据流会话**输出接线端至主VI中的另一个点对点数据流VI，以控制和监控数据流会话。

相关概念：

- 配合使用点对点数据流和FPGA终端
- 创建至点对点读取方FIFO的引用
- 创建点对点写入方FIFO的引用

同步FPGA和主机

FPGA VI与主控VI固有不同步。如需要在FPGA VI和主控VI间同步操作或传输数据，可使用基于中断或轮询的方法。需要最小化主机处理器的使用量时可使用基于中断的同步，需要较小的延迟时可使用基于轮询的同步。DMA FIFO是一种可用于在FPGA和主机间传输数据的特殊的基于轮询的同步。下表总结了上述同步方法的特性。

同步方法	延迟	主CPU使用量	常用于
基于中断	高	下限	数据记录
基于轮询	下限	高	控制，仿真
DMA FIFO	下限	下限	数据记录

基于中断的同步

中断为FPGA终端插入至主机的物理硬件数据线。可以使用中断通知主控VI事件，如数据变为可用、产生错误或任务完成。

使用基于中断的通信相对于使用轮询的通信的优势在于主控VI等待中断的同时可执行其他操作。但处理中断所需的系统开销也相应地增加了延时。

通过中断VI生成FPGA终端可用的32位逻辑中断。每个逻辑中断指出了产生中断的原因，并允许用户在软件中进行不同的处理。可将中断VI设置为等待直至主控VI通过连线**等待前清零**输入端确认FPGA终端的中断。在上述情况下，中断VI等待直至主控VI控制设备确认中断。

当FPGA终端的调用包含同步中断时请使用警告。当使用多于1个中断时，中断数据线用作共享资源，且将产生抖动。

基于轮询的同步

基于轮询的同步使用循环持续检查FPGA VI的状态，且当特定条件存在时执行动作。定时VI可用于确定轮询循环的频率。增加轮询循环的频率可降低同步的延时。降低轮询循环的频率可减少主机CPU的使用量。

直接内存访问(DMA)和同步

DMA FIFO是固有同步的。DMA引擎自动轮询DMA FIFO的状态，且当DMA FIFO包含指定数量的数据值时开始数据传输。但用户可指定设备从DMA FIFO获取数据的方式。具体来说，可以使用基于轮询的方法从DMA FIFO获取数据。

相关概念：

- 管理共享的资源
- DMA传输工作原理
- 设计主控VI在DMA应用中读取数据

在FPGA上生成中断

某些FPGA中断允许通过FPGA VI生成中断以通知事件的主控VI。例如，数据已准备就绪、产生错误或任务完成。如要判定终端是否支持中断，请访问**FPGA终端属性**对话框的**常规**页面的**终端信息**部分。

> **注：** 在FPGA VI的定时循环中不能使用“中断”VI。但在定时循环中可使用“设置事件发生”函数，在独立的While循环中可使用“等待事件发生”函数。然后可在While循环内使用“中断”VI，以在事件发生时生成中断。

按照下列步骤在FPGA VI中生成中断。

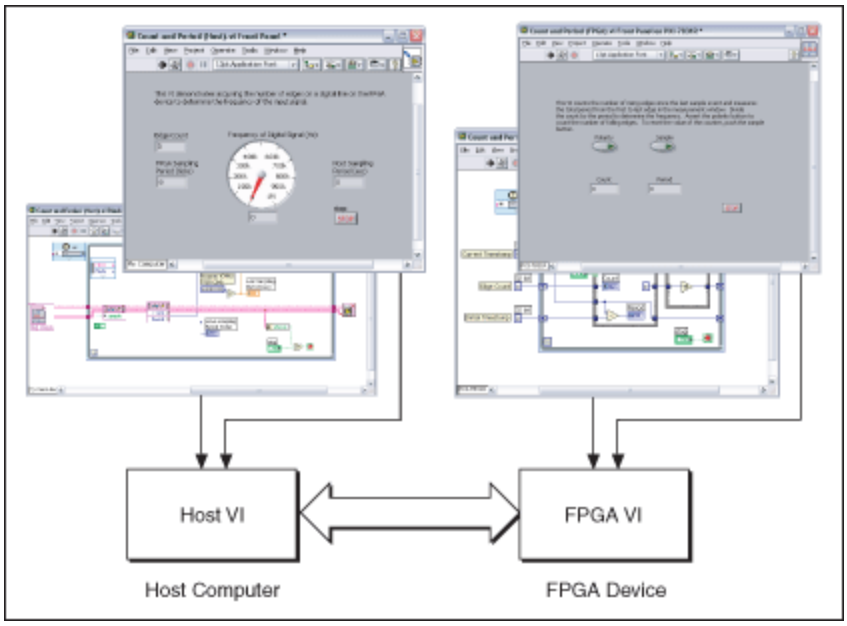
1. 添加中断VI至FPGA VI的程序框图中想要生成用于主控VI中断的数据流位置。
2. 右键单击“中断”VI的**IRQ编号**输入端并从快捷菜单中选择**创建»常量**。用户也可创建一个输入控件或连线另一个程序框图节点的输出端至“中断”VI的输入端。
3. 在**IRQ编号**输入端键入要用的逻辑中断的值。逻辑中断的值用于主机区分FPGA VI中设置的多个中断。如在FPGA VI中仅设置了一个中断，可使用任意逻辑中断编号。
4. 右键单击“中断”VI的**等待前清零**输入端并从快捷菜单中选择**创建»常量**。
5. 如要“中断”VI等待直至主控VI确认中断，设置**等待前清零**布尔常量的值设置为TRUE。如不需要“中断”VI等待直至主VI确认中断，设置**等待前清零**布尔常量的值设置为FALSE。

关于在FPGA接口中使用中断的详细信息，见使用中断同步FPGA VI和主控VI。

使用主控VI与FPGA终端通信

使用运行在主控计算机上的独立VI，可通过编程与FPGA VI交互。可编程式FPGA接口通信与交互式前面板通信不同，因为它需要用户创建一个主控VI及FPGA VI。

使用可编程式FPGA接口通信时，FPGA VI运行在FPGA终端上，主控VI运行在主控计算机上。如下列示意图所示。



使用主控VI在主控计算机和FPGA终端间发送信息可能源自下列原因：

- 处理超出FPGA设备限制的数据。
- 执行FPGA终端不支持的运算。例如，双精度或扩展精度浮点型算术。
- 通过FPGA终端创建一个多层应用，并用作大型系统的一个组件。
- 记录数据。
- 控制定时和数据传输顺序。
- 为FPGA VI创建一个测试台。

主控计算机可为基于Windows的计算机或RT终端。Windows操作系统和RT操作系统均支持FPGA接口函数。此外，可使用RT终端上的FPGA接口函数与FPGA终端通信，然后使用基于Windows的计算机与RT终端通信。

注： FPGA接口函数不能用于Linux操作系统。在Linux操作系统可使用FPGA Interface C API。

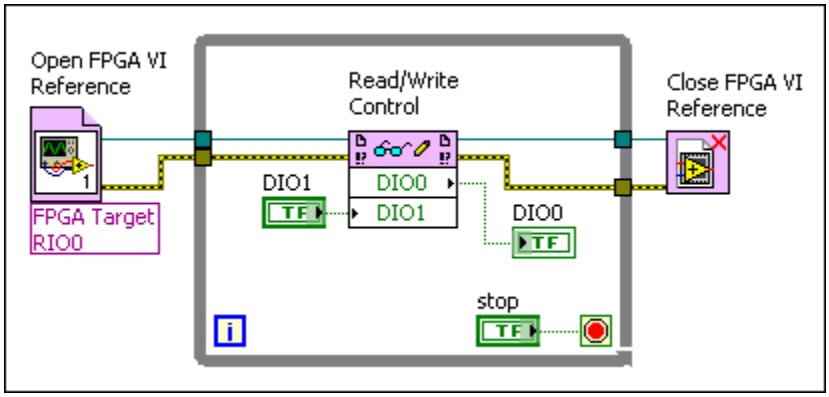
下列步骤为主控VI的常用编程顺序：

1. 打开至FPGA VI的引用，创建程序生成规范或位文件。
2. 使用FPGA接口函数（例如，读取/写入控件和调用方法函数）发送或接收数

据。

3. 使用关闭FPGA VI引用函数关闭FPGA引用。

下列程序框图给出了一个控制图中架构的简单主控VI。在该主控VI中，应用程序写入一个值至FPGA上的布尔输入控件**DIO1**，从FPGA上的布尔显示控件**DIO0**读取值。



While循环中代码的改变取决于应用及在FPGA和主机间传输数据的方式。

相关概念：

- 编译、下载和运行FPGA VI
- 通过主控计算机与FPGA终端通信
- 创建仿真I/O的自定义VI
- 使用前面板输入控件和显示控件传输数据

FPGA接口

2014年6月，371107L-0118

FPGA接口提供了编程功能及函数，帮助用户与运行在NI FPGA终端（例如，NI RIO设备）上的VI或比特文件通信。运行在FPGA终端上的VI称为FPGA VI。用于与FPGA VI或比特文件通信的VI称为主控VI。主控VI可运行在计算机上或RT终端上。

FPGA接口适用于FPGA终端驱动程序软件。如具有FPGA终端和相应的驱动程序软件，即可在主控VI中使用FPGA接口函数。无需安装LabVIEW FPGA模块以使用“FPGA接口”函数。如要开发FPGA VI，则需要安装FPGA模块。

在不带有FPGA模块的情况下使用LabVIEW FPGA接口

如带有FPGA终端，用户可在未安装LabVIEW FPGA模块的情况下，使用FPGA接口VI和函数与FPGA位文件交互。使用“打开FPGA VI引用”函数或“打开动态比特文件引用”函数打开指向FPGA位文件的引用。然后可使用其他FPGA接口函数（例如，读/写控件、调用方法和关闭FPGA VI引用）与FPGA位文件交互。

注： 只有在安装了LabVIEW FPGA模块的情况下可使用未编译的FPGA VI。

相关概念：

- 打开FPGA VI的引用、程序生成规范或比特文件

与FPGA VI通信

本节的主题介绍了使主控VI能够与FPGA VI通信的函数和程序。

打开FPGA VI的引用、程序生成规范或比特文件

您可以使用主控VI与在FPGA终端上运行的FPGA VI或比特文件进行通信。主控VI可运行在计算机上或RT终端上。每个主控VI必须打开FPGA VI引用、程序生成规范或运行在FPGA终端上的位文件。可打开指向任意FPGA VI的引用或与主控VI属于同一LabVIEW项目的程序生成规范。可打开指向项目内部或外部位文件的引用。

如要打开FPGA VI的引用，FPGA终端、FPGA VI和主控VI必须位于同一个LabVIEW项目中。如打开比特文件的引用，主控VI无需位于该项目中。

注： 如要在一个终端上打开不同FPGA VI或位文件的引用，请每次仅打开一个引用，且打开一个新的引用前应关闭上一个引用。在所有引用均关联同一个终端上的同一个FPGA VI或位文件时，可在一个终端上一次打开不止一个FPGA VI引用。

打开FPGA VI的引用或程序生成规范

按照下列步骤打开主控VI中的FPGA VI的引用或程序生成规范。如主控VI、FPGA终端、FPGA VI和程序生成规范位于同一个项目中可打开引用。如未安装LabVIEW FPGA模块，不能打开FPGA VI的引用或程序生成规范。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目或验证项目浏览器窗口中的FPGA终端。
3. 新建一个FPGA VI或验证位于**项目浏览器**窗口中FPGA终端下的要打开的引用的FPGA VI。
4. 新建一个主控VI或在项目中打开现有的主控VI。主控VI必须位于**项目浏览器**窗口中的**我的电脑**或RT终端下。
5. 在程序框图上添加“打开FPGA VI引用”函数。
6. （可选）从**项目浏览器**窗口拖曳要打开引用的FPGA VI至至“打开FPGA VI引用”函数。“打开FPGA VI引用”函数中将出现FPGA VI图标。如项目中的FPGA终端与物理终端相关联，终端名称和资源将出现在“打开FPGA VI引用”函数下。
7. （可选）右键单击“打开FPGA VI引用”函数，从快捷菜单中选择**配置打开FPGA VI引用**显示配置打开FPGA VI引用对话框。通过此对话框可选择程序生成规范和其他用于打开引用的选项。
8. （可选）连线输入控件或常量至“打开FPGA VI引用”函数的**资源名称**输入端，指定要运行FPGA VI的FPGA终端。

> **注：** 在主控VI中必须为每个“打开FPGA VI引用”函数添加和连线关闭FPGA VI引用函数。

打开比特文件的引用

按照下列步骤在主控VI中打开位文件的引用。主控VI无需位于项目中。如应用程序需要指向运行时位文件的引用，可使用“打开动态比特文件引用”函数替换“打开FPGA VI引用”函数。

1. 新建一个主控VI或打开一个现有的主控VI。如主控VI位于项目中，其必须位于**项目浏览器**窗口中的**我的电脑**或RT终端下。

2. 在程序框图上添加“打开FPGA VI引用”函数。
3. 右键单击“打开FPGA VI引用”函数，从快捷菜单选择**配置打开FPGA VI引用**。
4. 在**配置打开FPGA VI引用**对话框中选择**位文件**选项。
5. 浏览要在FPGA终端中打开的位文件。
6. （可选）使用**配置打开FPGA VI引用**对话框选择打开引用的其他选项。
7. 单击**OK**按钮。“打开FPGA VI引用”函数中将出现FPGA VI图标。“打开FPGA VI引用”函数的左上角将出现一个文件夹图标以提醒用户位文件。
8. 连线输入控件或常量至“打开FPGA VI引用”函数的**资源名称**输入端，指定要运行FPGA VI的FPGA终端。

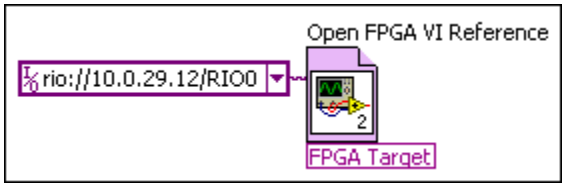
注： 在主控VI中必须为每个“打开FPGA VI引用”函数添加和连线关闭FPGA VI引用函数。

相关概念：

- 下载FPGA VI至FPGA终端
- 使用用于同一个终端的多个FPGA VI引用
- 从主控VI读取DMA FIFO
- 读取FPGA VI显示控件
- 使用中断同步FPGA VI和主控VI
- 在不带有FPGA模块的情况下使用LabVIEW FPGA接口
- 从主控VI写入DMA FIFO
- 写入FPGA VI输入控件

通过网络远程访问FPGA终端

使用打开FPGA VI引用函数可通过网络与FPGA终端上运行的FPGA VI或比特文件通信。如要通过网络访问FPGA终端，连线一个远程地址至打开FPGA VI引用函数的**资源名称**接线端，如下图所示。



在上述程序框图中，RIO设备I/O常量指定了一个远程FPGA终端。通过**FPGA终端属性**对话框或直接在FPGA接口VI的“打开FPGA VI引用”函数中指定一个远程终端。

提示 NI Measurement & Automation Explorer (MAX)检测远程FPGA终端并在其连接至的远程系统的**设备和接口**类别中显示其信息。

在FPGA终端属性对话框中指定一个远程FPGA终端

FPGA终端属性对话框指定一个远程FPGA终端时，远程FPGA终端出现在RIO设备I/O控件或主控VI常量的下拉菜单中。

按照下列步骤在**FPGA终端属性**对话框中通过网络指定一个FPGA终端。

1. 在**项目浏览器**窗口右键单击FPGA终端，从快捷菜单中选择**属性**，可打开“FPGA终端属性”对话框。
2. 在**常规**页面的**资源**文本框中输入以下字符串：
   `rio://`remote_target_computer`/`fpga_resource_name，其中remote_target_computer是计算机的IP地址或名称，fpga_resource_name是要访问的FPGA资源的名称。
3. 使用“打开FPGA VI引用”函数指定远程FPGA终端

在FPGA接口VI中指定远程FPGA终端

按照下列步骤配置“打开FPGA VI引用”函数，以通过网络访问FPGA终端。

1. 在程序框图上放置打开FPGA VI引用函数。
2. 切换至前面板窗口。
3. 在VI的前面板窗口放置一个RIO设备输入控件，位于I/O选板。
4. 使用下列方法之一指定远程FPGA终端：
   - 单击RIO设备输入控件的下拉箭头，从下拉菜单中选择一个远程FPGA终端。必须在“FPGA终端属性”对话框中指定远程FPGA终端，该选项才可用。

     提示 从下拉菜单中选择**浏览**，查找网络上的远程FPGA终端。

- 通过下列步骤手动输入远程FPGA终端的地址：
  a. 右键单击RIO设备I/O输入控件，从快捷菜单中选择**查找»接线端**。
  b. 右键单击程序框图上的RIO设备接线端，从快捷菜单中选择**转换为常量**。
  c. 在RIO设备常量的文本框中输入以下字符串：
  `rio://`remote_target_computer`/`fpga_resource_name，其中remote_target_computer是计算机的IP地址或名称，fpga_resource_name是要访问的FPGA资源的名称。

> **提示** 也可以单击RIO设备常量的下拉箭头，从快捷菜单中选择要访问的终端或选择**浏览**查找网络上的FPGA终端。

通过网络访问FPGA终端的最佳实践

远程访问FPGA终端时，请考虑下列规范。

- RIO服务器必须在主控系统上运行，运行在Windows系统中的VI才能通过网络连接访问RIO设备。在运行LabVIEW Real-Time的主控系统上（例如，CompactRIO控制器），RIO服务器将自动运行。在运行Windows的主控系统上（例如，NI PXI-7811R），必须手动开启RIO服务器。关于在Windows操作系统开启RIO服务器的相关信息，见`ni.com`的知识库。
- 访问远程FPGA终端时，避免对来自**FPGA VI引用输出**接线端的连线进行分支。连线分支后，LabVIEW不能确保在FPGA终端上并行但进行独立的运算。请创建使用2个“打开FPGA引用”VI实例的独立的引用。
- NI-RIO驱动程序确保用户不会丢失任何通过网络发送的数据，但其不能确保确定性或速度。网络传输速度可变，且通常要比所有硬件均为本地硬件的传输速度慢。

与在仿真模式执行的FPGA VI通信

下文列出了使用主控VI与在仿真模式使用仿真I/O执行的FPGA VI通信的注意事项：

- 必须在运行Windows操作系统的开发计算机上运行主控VI，以与FPGA VI通信。

如需使用仿真I/O，则不能在RT终端上运行主控VI。

- 如要基于FPGA VI执行的位置在主控VI上执行不同的代码，可使用配置为“获取FPGA VI执行模式”方法的调用方法函数。
- 此外，向上类型转换函数、动态FPGA接口转换函数及调用方法函数的中止、重置和下载方法不支持在仿真模式执行的FPGA VI。如使用上述函数或方法，主控VI将返回运行时错误。
- 主控VI或FPGA VI执行时，直接内存访问(DMA) FIFO有效。如两个个VI均停止执行，DMA FIFO将丢失全部的数据。
- 仅当FPGA VI执行时中断有效。如FPGA VI停止执行，所有中断数据将丢失且任何主机接口将立即等待返回的数据。
- 如使用调用方法函数读取DMA FIFO，函数可能频繁超时，因为FPGA VI在仿真模式上的执行速率可能低于在FPGA终端上执行速率。
- 如要使用关闭FPGA VI引用函数关闭引用、停止FPGA VI以及重置在仿真模式执行的FPGA VI，执行FPGA VI前必须关闭FPGA VI的前面板窗口。如要使用关闭FPGA VI引用函数关闭主机引用，但不重置在仿真模式执行的FPGA VI，执行FPGA VI前必须打开FPGA VI的前面板窗口。

使用中断同步FPGA VI和主控VI

某些FPGA中断允许通过FPGA VI生成中断以通知事件的主控VI。例如，数据已准备就绪、产生错误或任务完成。如要判定终端是否支持中断，请访问**FPGA终端属性**对话框的**常规**页面的**终端信息**部分。

等待和确认信号中断

按照下列步骤在主控VI中等待和通知信号中断

1. 打开至FPGA VI的引用或比特文件。
2. 在数据流中需要主控VI等待来自FPGA VI的中断的位置，添加调用方法函数至主控VI的程序框图。请确保连线**FPGA VI引用输入**输入端。
3. 右键单击“调用方法”函数，从快捷菜单中选择**方法»等待IRQ**。
4. 右键单击“调用方法”函数的**IRQ编号**输入端并从快捷菜单中选择**创建»常量**。或者创建输入控件。

5. 输入在FPGA VI中选中的逻辑中断的值。
6. 如要指定主控VI继续数据流前等待的最大中断时间，请连线**超时**输入端。默认情况下，主控VI不会等待中断发生且仅返回已由FPGA VI设置的中断。或者为常量连线-1，即无限等待。使用**超时**输入端时，可使用**超时**输出端判定主控VI是在发生超时还是接收到中断时继续数据流。如发生超时，**超时**输出端将返回TRUE。
7. 右键单击“调用方法”函数的**IRQ置有效**输出端并从快捷菜单中选择**创建»显示控件**。LabVIEW将创建一个数值显示控件。显示控件的值为–1表示未接收到中断。
8. 在需要主控VI确认来自FPGA VI的中断位置的数据流中，添加调用方法函数至主控VI的程序框图。如需要函数在仅当主控VI收到中断时执行，可添加“调用方法”函数至条件结构。如连线值为TRUE的布尔常量至中断VI的**等待前清零**输入端，请在数据流中需要中断VI停止等待的位置添加“调用方法”函数。请确保连线“调用方法”函数的**FPGA VI引用输入**的输入端。
9. 右键单击“调用方法”函数，从快捷菜单中选择**方法»确认IRQ**。使用确认IRQ方法确认等待IRQ方法返回的逻辑中断。
10. 将“等待IRQ”方法的**IRQ置有效**输出端连接至确认IRQ方法的**IRQ编号**输入端。

等待和确认多个中断

按照下列步骤等待和确认主控VI中的多个中断

1. 打开至FPGA VI的引用或比特文件。
2. 在数据流中需要主控VI等待来自FPGA VI的中断的位置，添加调用方法函数至主控VI的程序框图。请确保连线**FPGA VI引用输入**输入端。
3. 右键单击“调用方法”函数，从快捷菜单中选择**方法»等待IRQ**。
4. 在程序框图上放置一个数组常量。数组由索引框、元素框和可选标签组成。索引框位于左侧，元素框位于右侧。
5. 添加数值常量至数组。
6. 使用定位工具将数组常量展开为所需的中断数。
7. 输入逻辑中断的值。
8. 连线数组常量至**IRQ编号**输入端。
9. 如要指定主控VI继续数据流前等待的最大中断时间，请连线**超时**输入端。默认

情况下，主控VI不会等待中断发生且仅返回已由FPGA VI设置的中断。或者为常量连线-1，即无限等待。使用**超时**输入端时，可使用**超时**输出端判定主控VI是在发生超时还是接收到中断时继续数据流。如发生超时，**超时**输出端将返回TRUE。

10. 右键单击“调用方法”函数的**IRQ置有效**输出端并从快捷菜单中选择**创建»显示控件**。LabVIEW创建一个数组显示控件。**IRQ置有效**前面板显示控件显示FPGA终端置有效的终端数值。空数组表示未接收到终端。
11. 在数据流中需要主控VI确认来自FPGA VI的中断的位置，添加“调用方法”函数至主控VI的程序框图。如需要函数在仅当主控VI收到中断时执行，可添加“调用方法”函数至条件结构。如连线值为TRUE的布尔常量至中断VI的**等待前清零**输入端，请在数据流中需要中断VI停止等待的位置添加“调用方法”函数。请确保连线“调用方法”函数的**FPGA VI引用输入**的输入端。
12. 右键单击“调用方法”函数，从快捷菜单中选择**方法»确认IRQ**。使用“确认IRQ”方法确认“等待IRQ”方法返回的逻辑中断。
13. 将“等待IRQ”方法的**IRQ置有效**输出端连接至确认IRQ方法的**IRQ编号**输入端。

相关概念：

- 打开FPGA VI的引用、程序生成规范或比特文件

使用用于同一个终端的多个FPGA VI引用

如要在一个终端上打开不同FPGA VI或位文件的引用，请每次仅打开一个引用，且打开一个新的引用前应关闭上一个引用。

在所有引用均关联同一个终端上的同一个FPGA VI或位文件时，可在一个终端上一次打开不止一个FPGA VI引用。在主控VI中打开不止一个FPGA VI引用，用户可结构化主控VI，以使主控VI的不同部分可与FPGA VI的不同部分通信。例如，主控VI可访问输入控件和显示控件、逻辑中断和DMA通道。因此，可划分主控VI，以使用打开FPGA VI引用函数的不同实例访问这些项。使用多个FPGA VI引用的一种最安全的访问类型就是读取输入控件和显示控件。

注： 多个主控VI可访问同一个FPGA VI。但必须注意避免同时从同一个

FPGA VI进行写入和读取操作。

相关概念：

- 打开FPGA VI的引用、程序生成规范或比特文件

从主控VI读取DMA FIFO

按照下列步骤在主控VI中读取DMA FIFO。

1. 打开FPGA VI或比特文件的引用。

   注： 如要打开FPGA VI的引用，FPGA终端、FPGA VI和主控VI必须位于同一个LabVIEW项目中。如打开比特文件的引用，主控VI无需位于该项目中。如打开一个FPGA VI的引用，项目必须包含一个位于FPGA终端下的DMA FIFO项，且FPGA VI必须包含一个FIFO方法节点，该节点在程序框图上配置了写入方法，可写入DMA FIFO项。

2. 在数据流中需要主控VI读取DMA FIFO的位置，添加调用方法函数至主控VI的程序框图。确保读取DMA FIFO前，主控VI运行了FPGA VI。连线**FPGA VI引用输入**输入端。
3. 单击“调用方法”函数，从快捷菜单中选择**FIFO**»**读取**，其中**FIFO**是项目中FIFO项的名称。按照所需连线输入和输出端。
4. 将关闭FPGA VI引用函数添加至程序框图。
5. 连线调用节点的**FPGA VI引用输出**输出端至“关闭FPGA VI引用”函数的**FPGA VI引用输入**输入端。

注： 可仅使用带读取方法的“调用方法”函数读取DMA FIFO。如要通过主控VI对DMA FIFO进行更多的控制，还可使用“调用方法”函数的可选配置、开始和停止方法配置、开始和停止DMA FIFO。

相关概念：

- 打开FPGA VI的引用、程序生成规范或比特文件

从主控VI写入DMA FIFO

按照下列步骤在主控VI中写入DMA FIFO。

1. 打开FPGA VI或比特文件的引用。

> **注：** 如要打开FPGA VI的引用，FPGA终端、FPGA VI和主控VI必须位于同一个LabVIEW项目中。如打开比特文件的引用，主控VI无需位于该项目中。如打开一个FPGA VI的引用，项目必须包含位于FPGA终端下的DMA FIFO项，且FPGA VI必须包含配置了读取方法的FIFO方法节点。该方法用于在程序框图上读取DMA FIFO项。

2. 在数据流中需要主控VI写入DMA FIFO的位置，添加调用方法函数至主控VI的程序框图。连线**FPGA VI引用输入**输入端。
3. 单击调用方法函数，从快捷菜单中选择**FIFO»写入**，其中**FIFO**是项目中FIFO项的名称。按照所需连线输入和输出端。当数据被写入或当**超时**周期结束时，写入方法返回**剩余空元素**。
4. 将关闭FPGA VI引用函数添加至程序框图。
5. 连线调用节点的**FPGA VI引用输出**输出端至“关闭FPGA VI引用”函数的**FPGA VI引用输入**输入端。

> **注：** 可使用仅带写入方法的调用方法函数写入DMA FIFO。如要通过主控VI对DMA FIFO进行更多的控制，还可使用“调用方法”函数的可选配置、开始和停止方法配置、开始和停止DMA FIFO。

相关概念：

- 打开FPGA VI的引用、程序生成规范或比特文件

在主控VI上使用子VI

可使用与FPGA终端上的FPGA VI或位文件通信的主控VI中的子VI。如要引用子VI中的FPGA VI或位文件，可使用“打开FPGA VI引用”函数。“打开FPGA VI引用”函数可被配置为动态，此时即使引用位于不同的终端类型，子VI仍可与实现指定接口的FPGA引

用配合使用。例如，同一子VI可用于PCI和PXI终端。

按照下列步骤创建一个用于子VI的动态FPGA接口引用。

1. 在主控VI程序框图中，右键单击**打开FPGA VI引用**函数，从快捷菜单选择**配置打开FPGA VI引用**。此时将出现配置打开FPGA VI引用对话框。
2. 在**打开**选择程序生成规范、VI或位文件。
3. 勾选**动态模式**复选框。
4. 单击**确定**按钮关闭**配置打开FPGA VI引用**对话框。

使用动态FPGA接口引用

主机与通信硬件接口的相关部分兼容的情况下，能够创建可使用不同FPGA VI接口或比特文件的子VI。通信硬件接口包含下列：

- 输入控件和显示控件的名称和类型
- 名称、数据类型和DMA类型或点对点FIFO
- 名称、方向和终端范围方法的类型

即使引用指向位于不同终端类型的VI或位文件，上述子VI可与实现指定接口的任意FPGA引用配合使用。例如，同一子VI可用于PCI和PXI终端。

如要使FPGA引用为动态，勾选“配置打开FPGA VI引用”对话框的**动态模式**复选框。FPGA接口动态引用句柄常量和动态FPGA接口转换函数也可用于指定FPGA接口。如要创建一个子VI，可右键单击常量并从快捷菜单中选择**转换为输入控件**或**转换为显示控件**，转换常量为输入控件或显示控件。

编译时和运行时错误

使用FPGA接口动态模式时，LabVIEW返回编辑时和运行时错误。下列范例描述了可能产生编辑时或运行时错误的场景：

- 编辑时错误 – 使用调用方法函数配置FIFO，然后从项目中移除该FIFO时发生。调用方法函数将导致**运行**按钮断开。

- 运行时错误 – 使用动态FPGA接口强制转换函数转换引用，并使用读取/写入控件函数在不包含显示控件foo的引用上访问foo时发生错误。

确保动态模式引用不会断开下方数据流节点

配置带有动态模式选项的FPGA VI引用且其包含的元素多于子VI接口所需的元素数量时，可能出现强制转换问题并导致下方的数据流节点断开。如要确保退出子VI时的动态FPGA接口引用与FPGA接口引用进入子VI时的元素数量一致，必须执行下列操作之一：

- 分支连线输入至调用VI的引用。
- 直接连线子VI的输入引用至输出引用。

下载FPGA VI至FPGA终端

调用打开FPGA VI引用函数时，LabVIEW将自动下载编译FPGA VI至FPGA终端。如FPGA终端支持交互式前面板通信，在FPGA VI中单击**运行**按钮时，LabVIEW也会自动在FPGA终端上编译、下载和运行FPGA VI。如VI已位于FPGA终端或FPGA被保留用于其它用途时，LabVIEW不会下载FPGA VI。

在“项目浏览器”窗口右键单击FPGA VI并从快捷菜单中选择**下载**，可强制LabVIEW下载FPGA VI。关于终端可用下载选项的相关信息，见具体的FPGA终端硬件文档。

通过编程也可使LabVIEW从主控VI下载FPGA VI或比特文件至FPGA终端。

按照下列步骤通过编程下载FPGA VI或比特文件。

1. 打开FPGA VI或比特文件的引用。

   > **注：** 如要打开FPGA VI的引用，FPGA终端、FPGA VI和主控VI必须位于同一个LabVIEW项目中。如打开比特文件的引用，主控VI无需位于该项目中。

2. 将调用方法函数添加至程序框图。
3. 将“打开FPGA VI引用”函数的**FPGA VI引用输出**参数连线至“调用方法”函数的

FPGA VI引用输入参数。

4. 右键单击“调用方法”函数，从快捷菜单中选择**方法»下载**。

如要通过调用方法函数编程下载FPGA VI至FPGA终端，还必须通过编程在FPGA终端上运行带有调用方法函数的FPGA VI。添加另一个调用方法函数至程序框图，从调用方法函数快捷菜单中选择**方法»运行**，通过编程运行FPGA VI或比特文件。

相关概念：

- 编译、下载和运行FPGA VI
- 打开FPGA VI的引用、程序生成规范或比特文件

停止、中止和重置FPGA VI

如创建一个要再次通过主VI停止和运行的FPGA VI，建议将FPGA VI设计为可通过在FPGA VI中将控件设置为特定的值中止VI执行。此设计练习使用户能够控制FPGA VI的停止时间及FPGA VI停止时终端的输出值。通过编写可重入FPGA VI控件可由主VI停止FPGA VI。然后可使用带有“运行”方法的“调用方法”函数运行FPGA VI。如不需要控制FPGA终端输出，可使用带有“重置”方法的“调用方法”函数中止和恢复FPGA VI为默认状态。

如停止状态未设计为FPGA VI的一部分，可使用“重置”方法或“中止”方法重置和/或中止FPGA VI。“中止”方法不会重置FPGA VI为默认状态。默认状态将会对FPGA VI后续对主VI的“运行”方法的响应产生影响。

相关概念：

- 写入FPGA VI输入控件

在FPGA和主机间传递数据

如要在FPGA VI和主控VI间可编程式地传输数据，可使用下列方法：

- 可编程式前面板通信

- 直接内存访问(DMA)
- 用户定义I/O变量

下表总结并比较了上述方法。

数据传输方法	主控OS	吞吐率	调用系统开销	主CPU使用量	定时模型	同步
可编程式前面板通信	Windows, RT	下限	下限	高	用户定义	用户定义
直接内存访问	Windows, RT	高	高	下限	用户定义	自动
用户定义I/O变量	带有NI扫描引擎功能的RT系统	下限	下限	下限	NI扫描引擎	自动

通常，小型、频繁的数据传输可使用可编程式前面板通信，一次性传输大量数据时可使用DMA。使用用户定义I/O变量传输相于FPGA I/O数据集至RT主VI或从RT主VI接收数据。下表比较了使用下列传输方法的常见原因。

数据传输方法	常用于
可编程式前面板通信	• 从主控端发送配置参数或数据至FPGA • 从FPGA至主控端报告状态 • 在主控端和FPGA间发送命令 • 在主控端和FPGA间传输单点数据 • 创建用于FPGA VI的测试台
直接内存访问	• 在主控端和FPGA间传输大量的数据集 • 在主控端和FPGA间传输波形数据
用户定义I/O变量	• 与前面板通信执行相同的函数，NI扫描引擎使用定时和同步 • 传输相关的数据集

相关概念：

- 使用前面板输入控件和显示控件传输数据
- 使用直接内存访问传输数据
- 使用NI扫描引擎和变量传输数据

使用直接内存访问传输数据

直接内存访问(DMA)为在FPGA终端和主控计算机间基于FIFO传输数据的方法。DMA不包含主控处理器；因此其为在FPGA终端和主机间传输大型数据的最快方法。

DMA通信的优势

下表高亮显示了使用DMA通信在FPGA终端和主机间传输数据的优势：

- 释放了主机处理器，使其可在数据传输的过程中执行其他计算
- 限制前面板显示控件和输入控件的使用
- 传输数据数组时保存FPGA资源
- 自动同步主机和FPGA终端的数据传输

应用执行下列操作之一时可考虑使用DMA传输：

- 在FPGA终端和主控端传输波形数据
- 传输大型数据集合
- 数据记录
- 运行FPGA终端相比主控计算机更能有效处理的算法；例如，从多个输入通道获取平均值或求和值。

在应用中执行DMA通信

高电平时在FPGA应用中执行DMA通信包含下列步骤：

1. 判定FPGA终端是否支持DMA通信
2. 判定DMA是否为应用的最佳选择。请衡量DMA的优势及在FPGA终端和主控计算机间传输数据的其他选项。

3. 理解DMA工作原理。
4. 根据最佳时间设计和编程应用：
   - 评估多通道的需求。
   - 设计主控VI。
   - 避免缓冲错误。

相关概念：

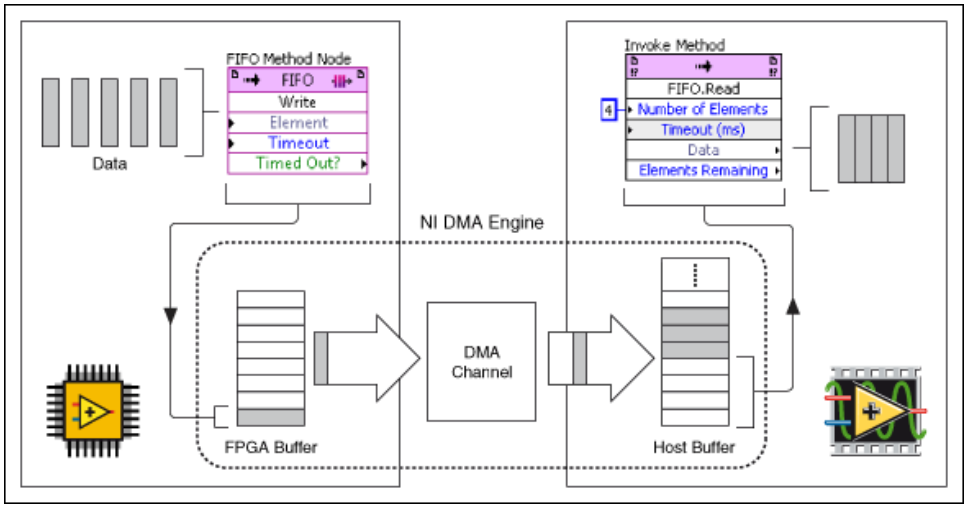
- 在FPGA和主机间传递数据
- 在设备或FIFO架构间使用FIFO传输数据
- 限制FPGA VI中顶层前面板对象的数量
- 在DMA应用中传输多通道数据
- 设计主控VI在DMA应用中读取数据
- 避免DMA缓冲错误

DMA传输工作原理

DMA通道包含两个FIFO缓冲区：一个位于主控计算机，另一个位于FPGA终端。创建DMA FIFO后，写入程序框图代码以从相应的缓冲区写入数据和读取数据。例如，如从FPGA传输数据至主机，用户在写入数据至缓存的FPGA上写入代码，同时也在从缓存读取数据的主机上写入代码。

> **注：** DMA通信是单向的。如要从主控计算机传输数据至FPGA终端，必需创建额外的FIFO并使用其他DMA通道。

由于DMA通信是基于FIFO的，数据传输每次操作一个元素。缓冲区内的第一个元素为传输至另一缓冲区的第一个元素。下列示意图为数据传输的范例。



在上述示意图中，FPGA VI获取数据并写入一个数据元素至DMA FIFO的FPGA缓存。主控VI每次从主机缓存中读取4个元素。

注： 设计使用DMA通信的应用程序时，正确设置缓冲区大小是可遵循的一种最佳实践。

相关概念：

- 同步FPGA和主机

判定FPGA终端是否支持DMA通信

按照下列步骤判定FPGA终端是否支持DMA通信。

1. 在项目浏览器窗口右键单击FPGA终端，选择**属性**。LabVIEW打开FPGA终端属性对话框。
2. 定位终端信息部分，其显示DMA通道的数量。

注：

- 关于终端支持和处理DMA通信的方式，见FPGA终端硬件文档。
- RIO扫描接口和NI扫描引擎需要两个DMA通道。在单个通道间交错数

> 据可节省通道数量。

相关概念：

- 在DMA应用中传输多通道数据

DMA应用的最佳实践

使用DMA通信时，请考虑下列最佳实践：

最佳实践	详细信息
关于终端如何处理DMA传输的信息，见FPGA终端硬件文档。	DMA的支持和动作随FPGA终端变化。上述区别将影响用户设计应用的方式。
正确设置缓冲区大小并检查上溢和下溢。	DMA缓冲的大小对应用的性能和鲁棒性将产生较大影响。
设计处理下列情况的应用： • 在FPGA终端上，FIFO方法节点的**超时?**显示控件返回TRUE。 • 在RT终端上，FIFO读取节点的**错误输出**显示控件返回代码–50400。	上述情况指示从其中一个缓冲区读取数据或写入数据时产生错误。如设计应用处理并避免上述情况，应用将具有更高的鲁棒性。
使用FIFO.写入方法从主控计算机传输数据至FPGA终端时，在尝试写入数据至缓冲区前，总是先读取**剩余空元素**显示控件的值。	用于下一个DMA操作的空元素数量是确保正确的。对于多数FPGA终端，该值对于后序操作有效且以每个后续写入的值的大小递减。但某些FPGA终端每次写入DMA通道后均重新计算该值。如在上述情况下写入通道，由于缓冲区已满，LabVIEW可能丢弃数据。此时写入操作也可能超时。
在主控计算机上使用FIFO.读取方法读取DMA数据时，在	使用轮询编程架构，在尝试读取缓冲区的数据前，先读取缓冲区中的剩余元素数量。

最佳实践	详细信息
尝试从缓冲区读取数据前，总是先读取**剩余元素**显示控件的值。	
配置可编程元素数量前停止FIFO。	如在FIFO运行时配置FIFO的深度，FIFO下次启动时重置并清空FIFO内的全部数据。
交错多通道数据时指定待写入缓冲区的元素数量为通道数量的整数倍。	该设计确保用户可从每个通道读取相同数量的数据。例如，如具有8个I/O通道，且在“FIFO属性”对话框中的**请求的元素数量**中指定80，缓冲区则为每个通道准备10个采样的空间。
在主机中同时读取或写入大量的数据。	DMA通信将导致计算性的系统开销；因此发送命令或少量的数据将浪费资源。

相关概念：

- 避免DMA缓冲错误
- 设计主控VI在DMA应用中读取数据
- 在DMA应用中传输多通道数据

避免DMA缓冲错误

DMA缓冲区可存储的元素数量（通常称为缓冲区深度）为设计鲁棒性应用的重要部分。缓冲区的元素数量对应用产生下列影响：

- 如缓冲区过小，读取和移除数据前缓冲区就被占满。即产生溢出。写入缓冲区的全部数据将丢失。
- 如缓冲区过大，它将占用FPGA和/或主控计算机的资源，这将降低应用的性能。
- 尝试从缓冲区读取超出缓冲区范围的元素时将产生下溢，且LabVIEW返回错误。

如要设计鲁棒性的应用，NI建议：

- 正确设置缓冲区的大小

- 设计应用程序检测并避免缓冲溢出
- 设计应用程序检测并避免缓冲下溢
- 设计应用程序避免在缓冲区累积并形成过期数据

下列场景提供了更多的策略信息。

正确设置缓冲区大小

下表介绍了组成DMA通道的两个缓存、如何配置缓存的大小、推荐的缓存大小及每个缓存的最大容量。

缓存的位置	如何配置	默认缓存容量	推荐缓存容量	最大缓存容量
FPGA终端	FIFO属性对话框的通用页面	1023个元素	1023个元素多数应用均无需更改FPGA缓存的大小。	取决于FPGA上的可用资源数及主机内存分配的字节数。如FPGA资源不足以处理指定的元素数量，FPGA VI编译将失败。关于DMA FIFO大小限制的详细信息，见指定FPGA终端的硬件文档。
主控计算机	“调用方法”函数的“FIFO.配置”方法	10000个元素或2倍FPGA FIFO缓冲的大小，取两者中的较大者	指定待读取或写入元素数量的5倍	取决于FPGA上的可用资源数及主机内存分配的字节数。如FPGA资源不足以处理指定的元素数量，FPGA VI编译将失败。关于DMA FIFO大小限制的详细信息，见指定FPGA终端的硬件文档。

检测和避免缓冲溢出

使用FIFO方法节点的“获取待写入元素数量”方法可检测是否产生了缓冲溢出。该方法返回用户可写入数据的缓冲区的可用空元素数量。如该值为零则表示缓冲区已满；如继续写入数据至缓冲区可能导致数据丢失。

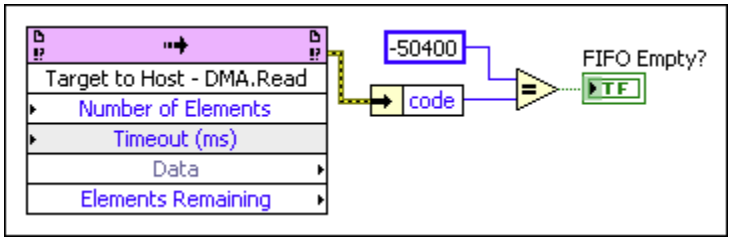
使用下列技术避免缓冲溢出：

- 降低用户写入缓冲区数据的速率。
- 增加主机上待读取的**请求的元素数量**。
- 增加主机读取数据的速率。
- 增加主机缓存容量、FPGA缓存容量或两者
- 降低主机CPU的处理负荷。CPU的速率及其当前的计算任务将降低从主机缓存至应用内存传输数据的速率。

检测和避免缓冲下溢

使用FIFO方法节点的“获取待读取元素数量”方法可检测是否产生了缓冲下溢。该方法返回可读取的缓存元素数量。如该数量小于用户从缓存中读取的元素数量，读取缓冲区可能导致缓冲区超时。

如要检测何时产生缓存下溢，可检查“FIFO.读取”方法是否发生错误–50400，如下图所示：



上述任一情况发生时，用户可能需要执行停止应用程序等类似的操作。产生溢出或下溢时，用户可使用配置为**或**的复合运算函数停止FPGA VI和主控VI的执行。

通过下列技术避免缓冲下溢：

- 增加“FIFO.读取”方法的的**超时**。
- 降低主机读取数据的速率。
- 通过减少连线至“FIFO.读取”函数的**元素数量**控件的值，减少VI从缓冲区读取的元素数量。

防止在缓冲区累积数据，形成过期数据

在FPGA终端执行VI、停止和重新启动VI时不会重置缓存。即下次运行FPGA VI时，数

据仍保存在缓冲区内。

通过下列技术清空缓存：

- 重置FPGA VI
- 刷新缓存以读取FIFO中的剩余元素。

相关概念：

- DMA应用的最佳实践
- 使用直接内存访问传输数据

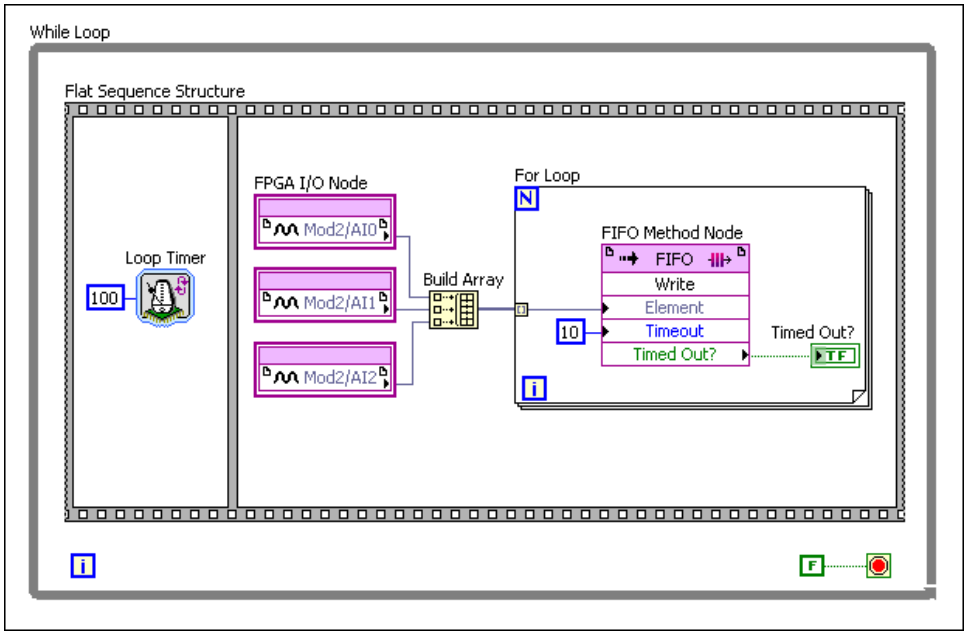
在DMA应用中传输多通道数据

如需要使用DMA传输FPGA终端的多个I/O通道的数据至主控计算机，建议写入数据至DMA缓存前先将每个通道的数据组合至单个数组。按照上述方式组合数据称为交错。由于传输交错的数组仅需要一个DMA通道，即释放了其他DMA通道用作其他任务，因此交错多通道数据很重要。从DMA缓存读取交错数据后，抽取数组以获取每个通道的数据。

注： 如所有通道的数据均为相同的数据类型，交错可发挥最佳性能。否则，LabVIEW写入缓冲区时可能强制转换数据，这将降低精度。

在FPGA终端上交错数据

下图为获取和交错数据的FPGA VI的示意图。



上图中的VI从3个输入通道获取数据：AI0、AI1和AI2。VI使用“创建数组”函数交错通道数据至单个数组。最后FIFO方法节点将该数组写入至DMA通道。DMA通道每次从FPGA终端传输数组的一个元素至主控计算机。

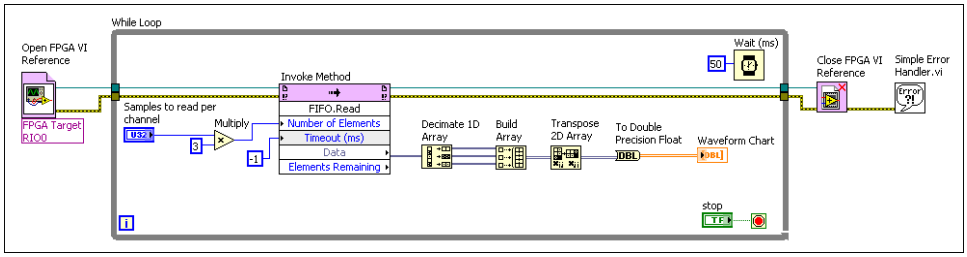
根据交错，数组包含下列元素：

数组元素	数据
0	来自AI0的采样1
1	来自AI1的采样1
2	来自AI2的采样1
3	来自AI0的采样2
4	来自AI1的采样2
5	来自AI2的采样2

注： 建议设置缓冲区的元素数量为用户交错的通道数量的整数倍。该设计确保缓冲区能够为每个通道存储相同数量的数据点。在该范例中，缓冲区有6个元素，即为每个通道提供两个元素的存储空间。

在主控计算机上抽取数据

下图为在主控计算机上读取数据的VI。



注意下图中的操作：

- 读取DMA通道时，必需指定要读取的**元素数量**。如要在该范例中计算该数量值，可使用每个通道的采样次数乘以通道总数。在该范例中，通道数量为3。
- “抽取一维数组”函数已被改变大小，以包含3个输出接线端；因此，该函数将交错数组拆分为3个独立的数组。第一个数组包含AI0的测量结果，第二个数组包含AI1的测量结果，第三个数组包含来自AI2的数组。

> **注：** 如还有第四个通道的数据，用户可重新改变抽取一维数组函数的大小，以包含第四个输出接线端。

- “创建数组”、“二维数组转置”和“转换为双精度浮点数”函数用于准备要在波形图表中显示的数据。用户可使用最适合应用的抽取数据的方法替换上述函数。

相关概念：

- DMA应用的最佳实践
- 判定FPGA终端是否支持DMA通信
- 使用直接内存访问传输数据

设计主控VI在DMA应用中读取数据

设计主控VI从DMA通道中读取数据是实现DMA应用的一个重要部分。

注： 本主题讨论的设计适用于从FPGA终端传输数据至主控计算机的应用。

选择设计

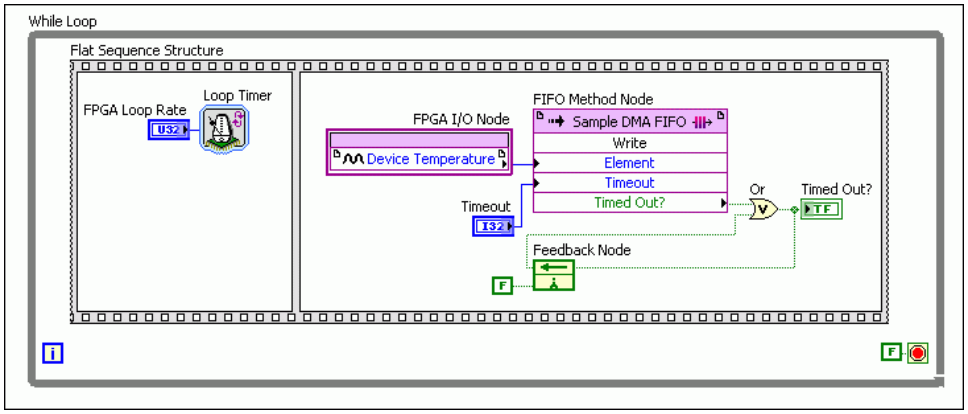
下表总结了某些有用的设计：

使用场景	设计	优点	缺点
发生下列一个或多个场景的应用： • FPGA以未知的速率获取数据。 • 数据采集的起始时间未知；例如，硬件设备触发器以未知或不规律的时间间隔采集数据。 • 执行其他任务前必须从缓冲区读取全部数据。	阻隔	• 最简的编程设计。 • 数据吞吐量高于任意轮询设计。	• 主控VI可读取全部**元素数量**或不读取。连线该数值至调用方法函数的“FIFO.读取”方法。 • 对于某些终端，等待**元素数量**变为可用时，主控VI不能执行其他任务。对于上述终端，如用户在实时系统上运行主控VI，主台式机系统可能暂时无法访问该实时系统。关于DMA限制的详细信息，见终端硬件文档。
数据记录应用，FPGA以未知的速率获取数据，但主控计算机以未知或不规律的速率读取数据。	轮询可变数量的元素	主控VI可在数据可用于读取时，立即读取任意数量的元素。	与块设计比较： • 读取数据耗时更长。 • 数据吞吐量更低。
在应用可继续执行前需要读取已知固定大小	轮询固	等待数据变为可用	• 主控VI可读取全部**元素数量**

使用场景	设计	优点	缺点
的数据，但等待数据变为可用时必须完成其他任务的应用程序。例如，假设一个将1024个元素处理为一个单帧数据的应用，该应用同时必须发送命令至另一个终端。定时轮询设计当数据可用时从缓冲区读取数据，数据不可用时发送必需的命令。 如FPGA需要获取未知或不规律速率的数据，该设计同样有用。	定数量的元素	时，主控VI可执行其他顺序任务。	或不读取。连线该数值至调用方法函数的“FIFO.读取”方法。 • 与块设计比较： ◦ 读取数据耗时更长。 ◦ 数据吞吐量更低。

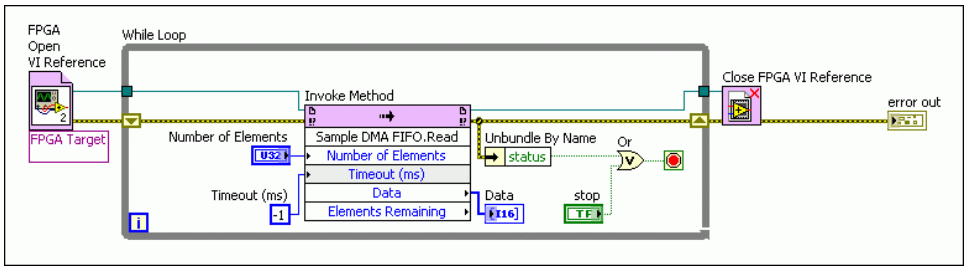
编程主控VI

考虑下列FPGA VI：

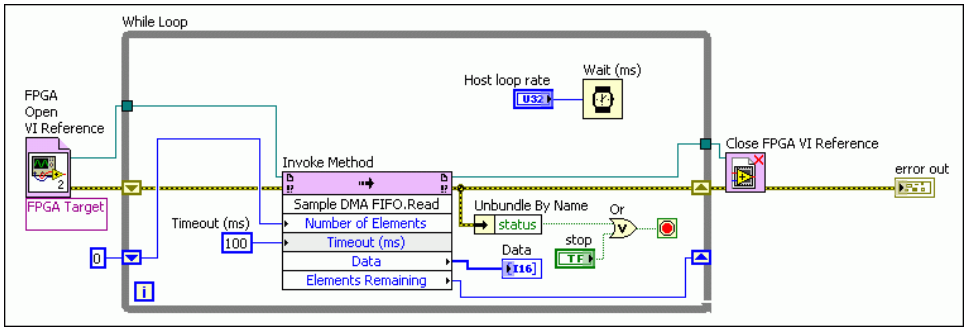


该VI通过下列方式表示数据采集：以预定义的速率测量FPGA设备的温度并写入该温度值至FPGA DMA缓存。下文显示了读取该数据的范例主控VI。

阻隔

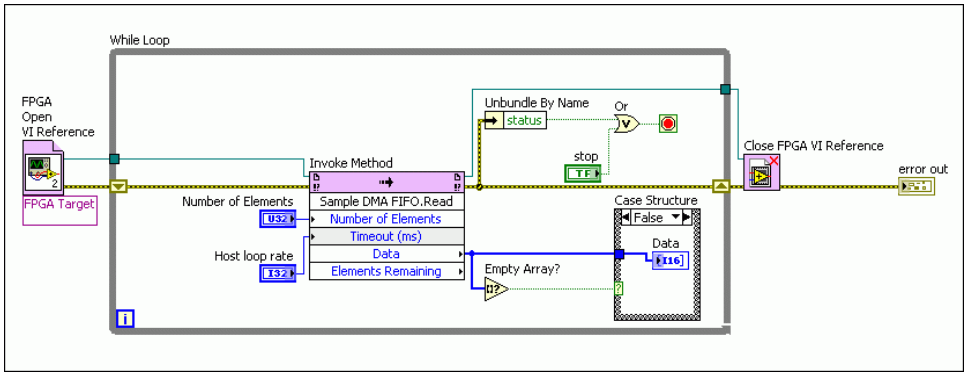


轮询可变数量的元素



注意，**元素数量**VI读取变化的方式基于上一次读取数据后**剩余元素**的数量。

轮询固定数量的元素



相关概念：

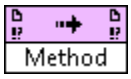
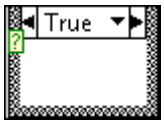
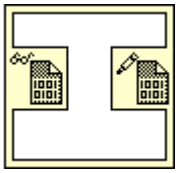
- DMA应用的最佳实践
- 同步FPGA和主机
- 使用直接内存访问传输数据

增强访问DMA FIFO的有效性

为了实现传统的DMA FIFO，LabVIEW使用多个数据副本在LabVIEW和设备驱动程序之间移动数据。这些额外的数据副本会消耗CPU周期，并限制DMA FIFO构建的应用程序的大小。对于大型的资源密集型的应用程序，可以考虑使用外部数据值参考，避免数据副本，降低CPU占用。

使用的对象

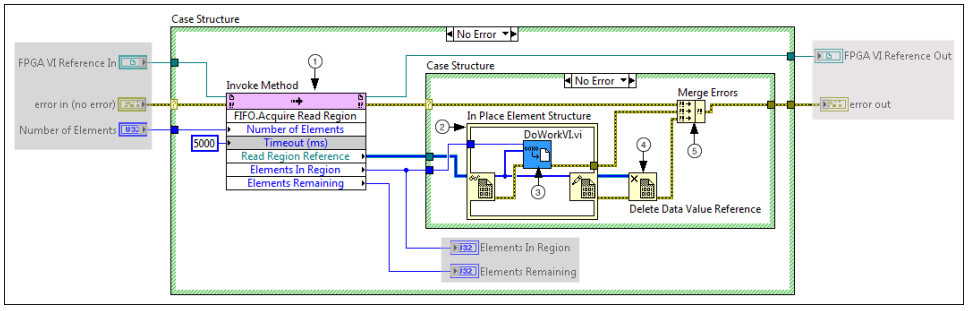
使用下列程序框图对象执行DMA传输，同时产生更少的数据副本：

调用方法函数	条件结构	元素同址操作结构	删除数据值引用	合并错误
				

操作步骤

创建下列程序框图，执行DMA传输，同时产生更少的数据副本。该程序框图使用了一个调用方法节点，节点配置为“采集读取区域”方法，从缓冲区采集读取区域，然后将引用通过元素同址操作结构传递至读取区域。元素同址操作结构中的子VI执行特定于应用程序的代码。在子VI执行后，删除数据值引用函数释放缓冲区上的范围。

根据您的实际需求自定义灰色部分的内容。



下面列出了上述程序框图的要点：

①	配置调用方法函数。根据应用程序的需要，选择采集写入区域或采集读取区域方法。
②	将读取区域或写入区域的引用连接至元素同址操作结构的数据值读取/写入节点。
③	加入定制的代码，实现程序的目标。
④	删除数据值引用。您必须删除数据值引用，释放缓冲区空间，使其他程序可继续访问缓冲区。
⑤	确保VI使用合并错误函数，按正确的顺序处理错误。必须先处理调用方法函数的输入错误，然后处理元素同址结构的输入错误。

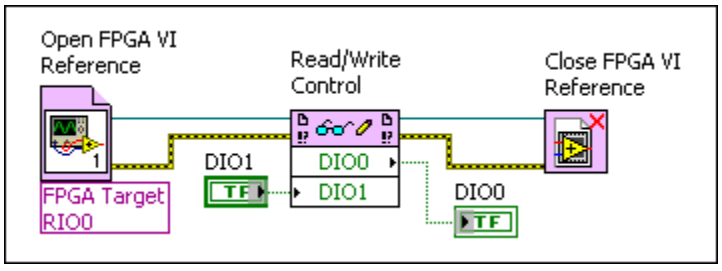
提示和疑难解答

在采集一个区域时，元素数量和区域中的元素并不一定相等。对于某些终端，内存中分配的字节数会被强制转换为某个终端有效的值，且为4096的倍数。对于这些目标，指定**元素数量**为4096的倍数，以避免缓冲溢出。关于有效FIFO深度的详细信息，请参考终端硬件文档。

如果缓冲区无法采集某个区域，则调用方法函数返回错误-61219。在大多数情况下，没有删除调用方法函数返回的数据值引用可引起该错误。如应用程序通过代码处理错误，请确保程序将检查错误-61219。

使用前面板输入控件和显示控件传输数据

在主控VI中使用读取/写入控件函数访问FPGA VI的前面板输入控件和显示控件，如下列程序框图所示。



注： 可编程前面板通信的支持随FPGA终端变化。更多信息见FPGA终端硬件的文档。

相对于其他在FPGA和主机间传输数据的方法来说，可编程式前面板通信的优势在于其低系统开销。尽管通过可编程式前面板通信不能实现高吞吐率，但每次对读取/写入控件函数的调用均以最小延时初始化数据传输。因此可编程前面板通信是小型、频繁数据传输的理想选择。

可编程式前面板通信的缺点是：该方法仅能传输存储在FPGA VI输入控件或显示控件中的最新数据。例如，如FPGA VI写入数据至显示控件的速率高于主控VI读取数据的速率，将产生数据丢失。且FPGA VI的每个输入控件或显示控件均占用FPGA资源。FPGA编程时，建议尽可能地减少FPGA VI中的前面板对象。另外，在FPGA VI和使用前面板输入控件和显示控件的主控VI间传输数据需要使用主机处理器资源。因此，数据传输的速率极大程度上取决于主机处理器的速度和可用率。较慢的处理器或处理器资源匮乏均会导致FPGA终端至主机的数据传输速率缓慢。

相关概念：

- 在FPGA和主机间传递数据
- 限制FPGA VI中顶层前面板对象的数量
- 使用主控VI与FPGA终端通信

写入FPGA VI输入控件

按照下列步骤从主控VI写入FPGA VI中的控件。

1. 打开FPGA VI或比特文件的引用。

   注： 如要打开FPGA VI的引用，FPGA终端、FPGA VI和主控VI必须位于同一个LabVIEW项目中。如打开比特文件的引用，主控VI无需位于该项目中。

2. 添加读取/写入控件函数到程序框图。注意，读取/写入控件函数包含一个**未选**

择输入。

3. 连线打开FPGA VI引用函数的**FPGA VI引用输出**输出端或打开动态比特文件引用函数的**比特文件引用输出**输出端至读取/写入控件函数的**FPGA VI引用输入**输入端。
4. 单击**未选择**输入端。快捷菜单将列出FPGA VI中的所有前面板输入控件和显示控件。

> **注：** 如未保存FPGA VI或不包含任何输入控件或显示控件，该列表为空。

5. 从快捷菜单中选择FPGA VI中可用的控件。注意，**未选择**输入端将发生变化以反映FPGA VI中输入控件的名称。

如要在FPGA VI中写入更多输入控件，右键单击读取/写入控件函数，从快捷菜单中选择**添加元素**，然后按照上述步骤自定义输入端。或者使用定位工具单击“读取/写入控件”函数的底部，向下拖曳边框以添加输入控件和显示控件。按照从上到下的顺序执行读取和写入操作。同样，如FPGA VI带有子VI，且用户要访问子VI内的输入控件和显示控件，必须连线子VI的输入控件和显示控件至用于“打开FPGA VI引用”函数中的FPGA VI的输入控件和显示控件。

> **注：** 也可写入FPGA VI显示控件。如要写入显示控件，可右键单击显示控件输出端并从快捷菜单中选择**转换为写入**。

相关概念：

- 停止、中止和重置FPGA VI
- 打开FPGA VI的引用、程序生成规范或比特文件

读取FPGA VI显示控件

按照下列步骤从主控VI读取FPGA VI中的显示控件。

1. 打开FPGA VI或比特文件的引用。

> **注：** 如要打开FPGA VI的引用，FPGA终端、FPGA VI和主控VI必须位于同一个LabVIEW项目中。如打开比特文件的引用，主控VI无需位于该项目中。

2. 添加读取/写入控件函数到程序框图。注意，读取/写入控件函数包含一个**未选择**输入。
3. 连线打开FPGA VI引用函数的**FPGA VI引用输出**参数或打开动态比特文件引用函数的**比特文件引用输出**参数至读取/写入控件函数的**FPGA VI引用输入**参数。
4. 单击**未选择**输入端。快捷菜单将列出**控件**子菜单中FPGA VI的全部前面板输入控件和显示控件。

> **注：** 如未保存FPGA VI或不包含任何输入控件或显示控件，该列表为空。

5. 从快捷菜单中选择FPGA VI中可用的显示控件。注意，**未选择**输入端将变化为输出端并反映FPGA VI中显示控件的名称。

如要读取FPGA VI中的更多显示控件，右键单击读取/写入控件函数，从快捷菜单中选择**添加元素**，然后按照上述步骤自定义输出端。或者使用定位工具单击“读取/写入控件”函数的底部，向下拖曳边框以添加输入控件和显示控件。按照从上到下的顺序执行读取和写入操作。同样，如FPGA VI带有子VI，且用户要访问子VI内的输入控件和显示控件，必须连线子VI的输入控件和显示控件至用于“打开FPGA VI引用”函数中的FPGA VI的输入控件和显示控件。

> **注：** 也可读取FPGA VI控件。如要读取控件，右键单击读取/写入控件函数的控件输入端，从快捷菜单中选择**转换为读取**。

相关概念：

- 打开FPGA VI的引用、程序生成规范或比特文件

使用NI扫描引擎和变量传输数据

通过可编程式前面板通信传输相干FPGA I/O数据集至RT主控VI时，必须创建同步FPGA VI与主控VI的程序框图代码。如FPGA终端支持NI扫描引擎，可使用NI扫描引擎同步数据传输。然后通过用户定义的I/O变量在FPGA VI和RT主控VI间传输数据。

注： I/O变量是一种共享变量，它使用NI扫描引擎对I/O数据进行单点访问。关于NI扫描引擎支持的详细信息，见指定FPGA终端的硬件文档。

使用NI扫描引擎减少用于访问和在FPGA I/O通道和RT主控VI间传输相干数据集的代码量。使用用户自定义I/O变量，发送数据至RT主控VI前和发送数据至FPGA VI后可在FPGA终端上处理数据。例如，创建一个执行下列步骤的应用：

1. 获取模拟I/O数据并对FPGA VI中的数据执行FFT
2. 传输处理后的数据至RT VI中的控制循环
3. 从RT控制循环传输输出的数据至FPGA，以用作物理I/O通道的输出

步骤2和3包含在FPGA VI和主控VI间传输数据的用户定义的I/O变量。

(CompactRIO)关于使用用户定义I/O变量的范例，见`labview\examples\CompactRIO\NI Scan Engine\Getting Started\User-Defined IO Variable - Basic\`目录下的`User-Defined IO Variable - Basic.lvproj`。

创建用户定义I/O变量

在**项目浏览器**窗口右键单击机箱项，从快捷菜单中选择**新建»用户定义变量**，新建一个I/O变量。但是，因为所有的I/O变量都不具有方向属性，必须将每个用户定义的I/O变量配置为**FPGA至主机**或**主机至FPGA**。

通过该方法创建的I/O变量将出现在标签为**用户定义变量**的容器中。

注： 每个机箱项仅能包含一个用户定义I/O变量的容器。但用户定义的I/

> O变量容器可包含多个用户定义的I/O变量。

用户定义I/O变量的说明

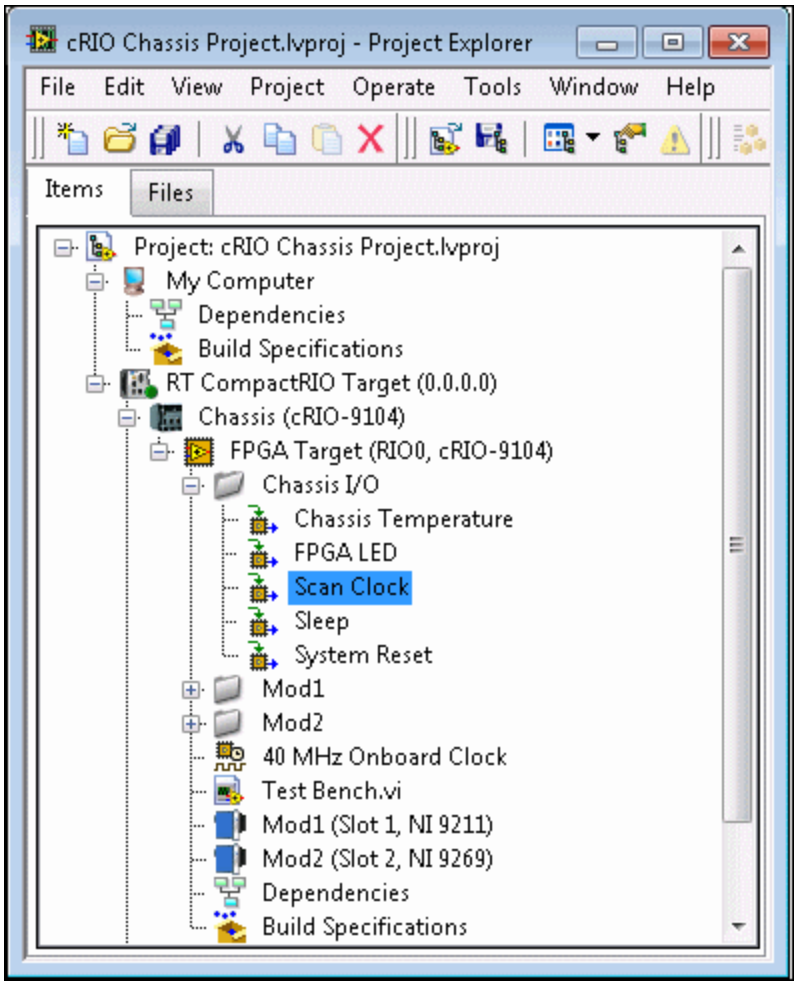
创建和使用用户定义I/O变量时请注意下列说明：

- 使用用户定义I/O变量前，必须打开FPGA VI的引用。
- 仅可在支持NI扫描引擎的FPGA终端上使用用户定义I/O变量。关于NI扫描引擎支持的详细信息见终端硬件文档。
- 用户定义I/O变量只支持扫描访问。不能直接访问用户定义I/O变量。
- 如添加用户定义I/O变量节点在FPGA VI程序框图上执行，必须设置FPGA VI的执行模式为FPGA终端。如要更改FPGA VI的执行模式，右键单击包含该VI的机箱项并选择**选择执行模式»FPGA终端**。如选择**仿真（仿真I/O）**或**仿真（实际I/O）**，且该FPGA VI包含用户定义I/O变量，**运行**按钮将显示断开且FPGA VI不能运行。
- 用户定义的I/O变量仅可用于运行在同一个机箱上的FPGA VI和RT VI间通信。如用户定义I/O变量已启用网络发布，可在任意RT VI或同一LabVIEW项目中的基于Windows的VI中使用该变量。例如，可使用网络发布I/O变量创建运行在Windows上的用户界面VI。

访问来自NI扫描引擎的定时信息

用户定义I/O变量基于来自NI扫描引擎的定时信息。通过添加Scan Clock I/O项至FPGA VI的程序框图可访问该定时信息。该I/O项传输来自扫描引擎的定时信息（例如，信号保持为高电平的FPGA时钟周期的数量）至FPGA VI。使用该定时信息设计应用程序，以确保在FPGA VI和RT主控VI间传输的数据集的一致性。

并非所有机箱均支持Scan Clock I/O项。如所用机箱不支持该项，它位于**项目浏览器**窗口**FPGA终端**的**机箱I/O**项下。下图为该项位置的示意图。


cRIO Chassis Project.lvproj - Project Explorer
File Edit View Project Operate Tools Window Help
Items
Files
Project: cRIO Chassis Project.lvproj
My Computer
Dependencies
Build Specifications
RT CompactRIO Target (0.0.0.0)
Chassis (cRIO-9104)
FPGA Target (RIO0, cRIO-9104)
Chassis I/O
Chassis Temperature
FPGA LED
Scan Clock
Sleep
System Reset
Mod1
Mod2
40 MHz Onboard Clock
Test Bench.vi
Mod1 (Slot 1, NI 9211)
Mod2 (Slot 2, NI 9269)
Dependencies
Build Specifications

调试FPGA VI

LabVIEW提供了几种调试整体或部分FPGA VI的方法。根据下表选择用于验证和调试的执行模式。

执行模式	验证应用性能	验证定时	验证HDL IP的集成	适用于单元测试	适用于组件测试	适用于系统测试
Windows PC	✓			✓		
仿真模式	✓				✓	
FPGA终端	✓	✓	✓		✓	✓
第三方仿真	✓	✓	✓		✓	

注： 在特定终端上还可使用交互式前面板通信调试FPGA VI。

如可对单元和组件类进行足够的调试和验证，能够减少系统级的验证。关于验证单元、组件和系统的规范见下文。

单位

单元是用户可创建的最基本的IP组件。它映射至指定的进程函数或算法，拆分单元并将其作为更小的功能单元进行测试的意义不大。组成单元的代码具有下列一个或多个特性：

- 代码不包含任何I/O、数据通信或终端资源
- 代码不包含并行运行或以不同速率执行的循环

- 代码本身可提供某些已知输入，测试预期的输出
- 代码不依赖于显式传输或时间控制
- 代码可被用作子VI，可在设计的其他部分进行重用

组件

组件为更复杂的逻辑组成，它依赖于系统定时。组件为模块化，通常用于完成明确的任务或目标。一个FPGA VI通常可被拆分为多个组件。此类组件验证确保组件集成至系统时，可实现预期交互。及可在配置整个系统前，确保子组件与I/O或主机VI交互正常。

系统

系统级组件可被视为是最顶层的组件，由顶层FPGA VI及通过CLIP导入的HDL IP表示。系统通常包含多个While循环或单周期定时循环。系统接口直接开放给主应用程序，因此验证测试与运行主应用程序类似，或包含主应用程序。系统级验证需要使用主机接口API，及连接全部实际I/O信号至系统。

相关概念：

- 使用第三方仿真器调试FPGA VI

定时冲突的疑难解答

用户可使用下列策略解决定时冲突分析窗口中显示的定时冲突。

- 如终端支持Xilinx选项页面，更改部分编译选项。
- 缩短较长的编译路径。
- 使用流水线。
- 减少嵌套条件结构的数量。
- 使用较小的数据类型。
- 重新编译FPGA VI。由于编译过程非确定性的映射FPGA VI至FPGA，每次编译FPGA VI时不会产生相同的结果。因此，如FPGA VI与所需的时钟速率仅相差几个

纳秒，重新编译FPGA VI即可修复定时冲突。
- 降低应用的时钟速率。
- 从独立于程序框图上的其他节点运行的单周期定时循环内部移除隐式启用信号。该策略主要用于大型设计。

相关概念：

- 更改顶层FPGA终端的时钟速率

使用仿真模式调试FPGA VI

编译FPGA VI可能耗时几分钟或几个小时。但编译VI前，可在仿真模式下使用仿真I/O测试FPGA VI的逻辑。使用该测试模式时，LabVIEW生成随机数据以用于输入，或使用用户创建的自定义VI来提供I/O。FPGA桌面执行节点可用于与选中的FPGA资源通信及调试FPGA设计。在某些FPGA终端上，可在仿真模式使用实际I/O执行FPGA VI。

在开发计算机上仿真FPGA VI时，可使用全部传统LabVIEW 调试技术（例如，探针、高亮显示执行、断点和单步执行），还可以使用专用于FPGA主控VI环境的调试技术。

提示 建议编译FPGA设计前，先在软件中进行调试和验证以避免不必要的编译过程。

如要在仿真模式，通过仿真I/O调试FPGA设计，请选择下列选项：

- **采样探针**－通过此选项检查VI运行时连线上的即时值，和查看信号数据随时间的变化。
- **FPGA桌面执行节点**－需要测试FPGA VI的独立设计元素或依赖于支持功能的设计的系统级测试时，选择此选项。支持的功能包括前面板输入控件和显示控件、I/O资源及使用仿真时间的资源。关于使用仿真时间的功能列表见“主机的仿真时间”章节。
- **自定义VI测试台**－需要创建来自特定输入的数据或监控输出时，使用此选项。

采样探针

在主控VI或FPGA VI中使用采样探针，查看VI运行时连线上的即时值和查看信号数据随时间的变化。例如，需要在单周期定时循环内调试信号。在主控VI中使用采样探针时，必须使用用于探针的采样源。

不能使用**采样探针监测窗口**更改数据。探针对VI的运行方式不会产生影响。

FPGA桌面执行节点

使用测试台调试FPGA VI前，可首先使用模拟I/O在模拟模式中测试VI逻辑，无需通过运行FPGA VI进行编译。该测试方法可节省编译时间，便于重复测试并减少创建用于其他调试的测试台所需的修改次数。

FPGA桌面执行节点在仿真模式使用仿真I/O运行FPGA VI，运行时间为指定数量的时钟滴答。使用FPGA桌面执行节点测试包含用户开发IP的单个循环，或测试包含多个循环的整个FPGA应用，应用中的循环以不同的时钟速率并行运行。使用该节点与选中的FPGA资源通信，并调试FPGA设计。

注： 如要连续仿真代码，必须将LabVIEW FPGA代码放置在While循环内。

自定义VI测试台

在LabVIEW中创建一个仿真I/O的自定义VI，用作FPGA VI的测试台。自定义VI可用作测试台，以创建来自指定输入端的数据或监控输出端。每次FPGA VI通过调用FPGA I/O节点读取输入值时，LabVIEW使用用户指定的自定义VI提供数据。通过FPGA I/O的自定义VI创建用于测试的可重复性场景，且使其能够更改指定输入端的数据。FPGA I/O的自定义VI也可用于监视来自FPGA I/O节点的输出端。

（状态图）在开发计算机上可调试用于FPGA终端的状态图。

理解仿真时间

如使用特定FPGA资源且在仿真模式使用仿真I/O执行FPGA VI，资源使用仿真时间而

不是实际时间。仿真时间可能比实际时间快，具体取决于仿真时发生的事件数量。例如，添加“等待（仿真时间）”VI至程序框图并将延时设置为1000 ms，LabVIEW不会尝试延时实际时间的一秒钟。LabVIEW执行仿真的下一个计划动作前，将延时必需的时间。

下列资源在主机上使用仿真时间：

- While循环
- 单周期定时循环
- 等待（仿真时间）VI
- 循环定时器Express VI
- 时间计数器Express VI
- FIFO（DMA FIFO除外）
- 等待事件发生超时（时钟滴答）函数
- **清零前等待**为TRUE时中断VI

相关概念：

- 清除FPGA FIFO
- 在FPGA终端上调试FPGA VI

使用FPGA桌面执行节点调试

使用FPGA桌面执行节点测试包含用户开发IP的单个循环，或测试包含多个循环的整个FPGA应用程序，其中的循环使用不同的时钟速率并行运行。由于FPGA桌面执行节点以仿真模式执行FPGA VI，可为FPGA VI开发包含终端资源（例如，I/O和存储器项）的测试。

注： 如要连续仿真代码，必须将LabVIEW FPGA代码放置在While循环内。

按照下列步骤使用FPGA桌面执行节点调试FPGA VI：

1. 在“项目浏览器”窗口，右键单击FPGA终端并选择**属性**，打开“FPGA终端属性”对

话框。

> **注：** 也可以右键单击FPGA终端，选择**选择执行模式»仿真（仿真I/O）**。

2. 在**执行模式**页面，选择**仿真**。
3. 在下拉菜单中选择**使用仿真I/O**。
4. 在主控VI中，添加FPGA桌面执行节点至程序框图。
5. 配置FPGA桌面执行节点。LabVIEW为用户指定的FPGA资源创建程序框图输入或输出。选择要调试的资源。
6. 单击**确定**完成配置FPGA桌面执行节点。
7. 运行主控VI。当前可使用标准LabVIEW调试工具分析FPGA桌面执行节点返回的仿真数据。

使用采样探针

使用主机VI或FPGA VI中的采样探针检查VI运行时连线上的即时值，和查看信号数据随时间的变化。例如，使用采样探针在单周期定时循环内调试VI，或在要用于单周期定时循环代码中调试VI。通过采样探针，能够在波形图上可视化多个信号，并能比较每个信号值各周期循环的变化。

> **注：** 采样探针仅支持标量、非簇的数据类型。例如，定点数、布尔值和整型。

使用**采样探针监视窗口**查看采样探针的数据。不能使用**采样探针监测窗口**更改数据。采样探针对VI的运行方式不会产生影响。

采样源

采样源是LabVIEW读取或采样相关探针数据时的判定信号。采样源确保采样探针相对于其他与源关联的采样探针，在正确的时间进行更新。对于用户可指定的采样源的数量没有限制。全部与给定采样源相关的探针显示在**采样探针监视窗口**的同一波形图中。

下表显示了主控VI和FPGA VI中采样源的区别：

注： 在FPGA VI中，LabVIEW仅支持在单周期定时循环内使用采样探针。位于单周期定时循环外的探针相对于定时来说，未必精确。

注： 由于LabVIEW在指定为采样源的循环执行结束后报告探针的值，因此在一次计数内，可能产生多个指定探针的值。例如，如该探针位于For循环内或被LabVIEW调用多次的子VI内，一次计数内可能生成多个探针值。**采样探针监测窗口**仅显示探针的最近一次数据。

在主控VI中指定采样源

按照下列步骤在主控VI的程序框图上设置用作采样源的循环：

1. 在以下循环中选择一个添加至程序框图。对于LoopWhile循环
2. 右键单击循环边框，从快捷菜单中选择**标记为探针采样源**，打开**采样探针监视窗口**。LabVIEW自动在**采样探针监测窗口**中列出和编号采样源，并在循环左上角的探针符号上显示相同的编号。

在FPGA VI中指定采样源

创建第一个采样探针时，LabVIEW将使用FPGA仿真时间自动创建**FPGA**采样源。该采样源不与程序框图中的任意结构相关联，但仅限在单周期定时循环内使用。

在程序框图上创建采样探针

按照下列步骤，在程序框图上添加一个采样探针：

1. 右键单击连线并选择**采样探针»x**，其中x是采样源的名称。
   - FPGA VI：显示**采样探针监视窗口**。
   - 主控VI：**采样探针监视窗口**已打开。

   LabVIEW在**采样探针监视窗口**中自动列出探针并为这些探针编号，窗口显示的值与探针符号所在连线的值相同。

2. （可选）可在其他连线或对象上放置更多探针，查看其数据变化。如有需要，可移动**采样探针监视窗口**。
3. 运行VI。**采样探针监视窗口**右边的波形图上显示流经连线的数据。
4. （可选）右键单击与探针相关的连线，从快捷菜单中选择**查找探针**，在**采样探针监视窗口**中找到探针。**采样探针监视窗口**关联至选中的探针。
5. 将**采样探针监视窗口**出现的值与期望值进行比较。
6. 在**采样探针监视窗口**选中采样探针，单击**采样探针监视窗口**工具栏中的**删除选中的探针**可删除采样探针。单击**删除全部**可删除当前显示的采样源及全部相关探针。关闭**采样探针监视窗口**或程序框图时，将自动关闭所有探针。

在主机上运行FPGA VI

完成下列步骤在主控计算机上运行FPGA VI。

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 新建一个VI或打开FPGA终端下的现有VI。

> **注：** 如FPGA VI使用I/O变量，用户不能在主控计算机上执行FPGA VI。如在主控计算机的FPGA VI的程序框图上放置了一个I/O变量节点，LabVIEW将断开**运行**按钮。

4. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**属性**。此时将显示**FPGA终端属性**对话框。
5. 在**类别**列表中选择**执行模式**，显示**执行模式**页面。
6. 选择下列选项之一：
   - **仿真**－选择该选项时，从下拉菜单中选择**使用仿真I/O**或**将自定义VI用于FPGA I/O**。

     > **注：** 选择**仿真I/O**与CLIP不兼容。

   - **仿真（实际I/O）**－某些FPGA终端不支持该选项。
7. 如在上一步中选择**将自定义VI用于FPGA I/O**，请指定自定义VI的路径或使用**VI路**

径控件**基于模板新建VI**。LabVIEW在FPGA VI程序框图上的FPGA I/O节点、FPGA I/O属性节点或FPGA I/O方法节点执行时调用指定VI。

8. 单击**OK**按钮。
9. 在FPGA VI上单击**运行**按钮。

提示 在**项目浏览器**窗口右键单击FPGA终端，从快捷菜单中的**执行模式**选择某一选项。

在主控计算机上运行FPGA VI的考虑因素

主控VI可用于与在仿真模式使用仿真I/O执行的FPGA VI通信。但使用主控VI时必须考虑特定的限制。

如在单周期定时循环内包含了数字I/O资源，对应于单周期定时循环的每次计数，每个同步寄存器均将引入延迟。在某些情况下，FPGA外部的延迟可能对系统影响较大。如LabVIEW框图和FPGA间延时的精确模型对于测试应用逻辑（通过在主控计算机上运行FPGA VI）非常重要，可将用于I/O的仿真数据的延时设置为对I/O节点的调用数量，即等于同步寄存器的数量。

验证在FPGA终端编译的FPGA VI

使用测试台调试FPGA VI时，用户在开发计算机上执行FPGA VI多次。但在编译FPGA终端的FPGA VI时需考虑特殊的限制。除测试FPGA VI逻辑外，必须考虑FPGA VI是否符合全部FPGA设计的限制条件。设计的限制条件包括应用是否占用了多余可用的FPGA资源的资源，以及逻辑是否符合FPGA时钟设置的定时限制。

建议周期性编译FPGA VI，以确保FPGA VI满足所有上述条件。此外，可使用程序生成规范快捷菜单中的命令生成中间文件和预估FPGA资源的使用量。中间文件的生成在整个过程中为相对较短的过程，且通常其将大量报错。

创建仿真I/O的自定义VI

测试台是为被测单元提供激励并读取返回的结果以验证设计的自定义VI。此时被测单元为FPGA VI。测试台通过或失败取决于响应。简单的测试台由两部分组成：

- **激励**—数据进入被测组件
- **响应**—被测组件返回数据

在LabVIEW中创建一个仿真I/O的自定义VI，用作FPGA VI的测试台。自定义VI可用作测试台，以创建来自指定输入端的数据或监控输出端。例如，如指定组件的预期输入为噪声正弦波，用户可使用自定义VI创建噪声正弦波。每次FPGA VI通过调用FPGA I/O节点读取输入值时，LabVIEW使用用户指定的自定义VI提供数据。通过FPGA I/O的自定义VI创建用于测试的可重复性场景，且使其能够更改指定输入端的数据。例如，如果需要使用方波而不是正弦波，用户可修改自定义VI。FPGA I/O的自定义VI也可用于监视来自FPGA I/O节点的输出端。例如，如组件的预期输出为正弦波且正弦波的频率为输出的重要组成，用户可创建一个自定义VI监控FPGA I/O节点的输出，以验证频率是否正确。

使用测试台调整FPGA VI前，可使用仿真I/O在仿真模式执行FPGA VI来测试VI的逻辑，此时无需编译VI。此操作节省了编译时间、使测试易于重现并降低了创建额外调试的测试台所需的改动数量。

创建自定义VI的测试台

按照下列步骤创建基于模板的自定义VI测试台。

1. 在**项目浏览器**窗口，右键单击FPGA终端显示**FPGA终端属性**对话框。
2. 在**执行模式**页面，选择**仿真**。
3. 在下拉菜单中，选择**将自定义VI用于FPGA I/O**。LabVIEW在该菜单下显示**VI路径**文本框。
4. 单击**基于模板新建VI**按钮。
5. 命名测试台并将其保存在`user.lib`目录下。文件路径将显示在**VI路径**文本框中。
6. 单击**确定**按钮添加自定义VI至项目。

关于创建和使用自定义VI测试台的范例，见教程：创建测试台。

相关概念：

- 交互式前面板通信
- 使用主控VI与FPGA终端通信

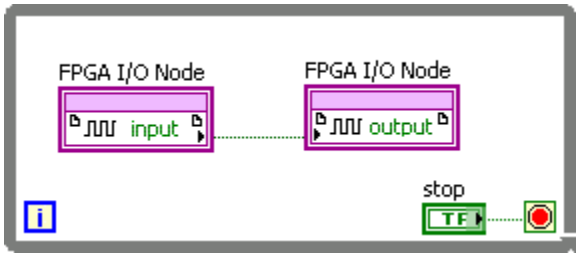
教程：创建测试台

本主题中的教程介绍了为一个简单FPGA VI创建测试台的步骤，该VI用于值的取反。尽管该测试台中的逻辑可能与您测试的FPGA VI中的逻辑不同，本指南将列出通用的步骤。指南假设用户已经掌握了创建VI的知识。

创建一个用于测试的FPGA VI

完成下列步骤以创建一个用于在本指南中进行测试的FPGA VI。

1. 创建一个LabVIEW项目，并将项目保存为`inverter.lvproj`。
2. 添加FPGA终端至项目。
3. 右键单击FPGA终端，从快捷菜单选择**新建»VI**，新建一个FPGA VI。
4. 将该FPGA VI另存为`Inverter.vi`。
5. 分别创建两个名为**输入**和**输出**的数字FPGA I/O项。
6. 从**项目浏览器**窗口拖拽FPGA I/O项至Inverter VI的程序框图，以创建相应的FPGA I/O节点。
7. 右键单击FPGA I/O节点的**输出**，从快捷菜单中选择**转换为写入**。
8. 将FPGA I/O节点放置在While循环内然后连线所有节点，如下列程序框图所示：



Inverter VI的作用是读取数字线的值然后取反，并将取反后的值写入另一数字线。注意该程序框图中包含一个错误。程序框图在两个FPGA I/O节点间缺少了一个非函数。

在开发计算机上使用随机数据测试FPGA VI。

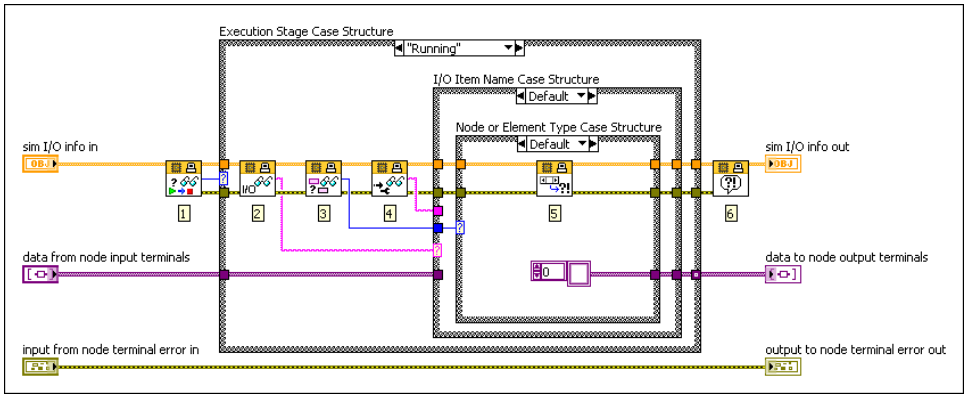
按照下列步骤使用随机数据作为输入，测试FPGA VI的逻辑。

1. 右键单击FPGA终端，从快捷菜单中选择**属性**。
2. 在**FPGA终端属性**对话框中，从**类别**列表中选择**执行模式**，显示**执行模式**页面。
3. 在**仿真**下拉菜单中，选择**使用仿真I/O**。
4. 单击**OK**按钮。
5. 在FPGA VI程序框图上，右键单击连接两个FPGA I/O节点的连线，从快捷菜单中选择**探头**，新建一个探头。
6. 运行FPGA VI。注意探针随机显示TRUE或FALSE的值。
7. 停止运行FPGA VI。

在开发计算机上使用自定义VI测试FPGA VI。

按照下列步骤使用自定义VI测试台测试FPGA VI。

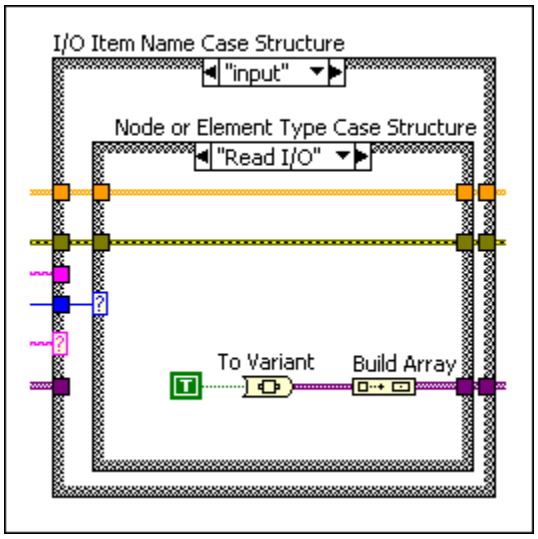
1. 在**FPGA终端属性**对话框的**执行模式**页面，从**仿真**下拉菜单中选择**将自定义VI用于FPGA I/O**。
2. 单击**基于模板新建VI**按钮，将自定义VI另存为`Simulated IO for Inverter.vi`。
3. 单击对话框中的**添加**按钮，对话框将提示您是否将该VI添加至当前项目。注意，LabVIEW将该自定义VI添加在了项目中的**我的电脑**下。
4. 打开"Simulated IO for Inverter"VI的程序框图，阅读VI中的注释，然后删除注释以降低程序框图的杂乱。"Simulated IO for Inverter"VI的程序框图如下图所示：



5. 右键单击**I/O项名称**条件结构，从快捷菜单中选择**复制分支**。
6. 在选择器标签中键入`input`。FPGA VI调用**输入**I/O项的FPGA I/O节点时，**输入**条件结构会执行。

> **注：** 如果使用CLIP将数据传输至FPGA VI，必须在CLIP I/O名称的选择器标签中额外添加一个反斜杠。例如，如果CLIP I/O项是**CLIP\Port A**，则必须在选择器标签中输入`CLIP\\Port A`。第一个反斜杠用作名称中第二个反斜杠的转义符。

7. 右键单击**节点或元素类型**条件结构，从快捷菜单中选择**复制分支**。注意此时已选中**读取I/O**条件分支。该条件分支用于配置的FPGA I/O节点读取**输入**I/O项。
8. 从该条件分支中删除“报告支持的错误”VI，穿过该条件结构连线**仿真I/O信息输入**和**错误输出**。
9. 使用真常量、“转换为变体”函数和“创建数组”函数替换空的数组，如下列程序框图所示：



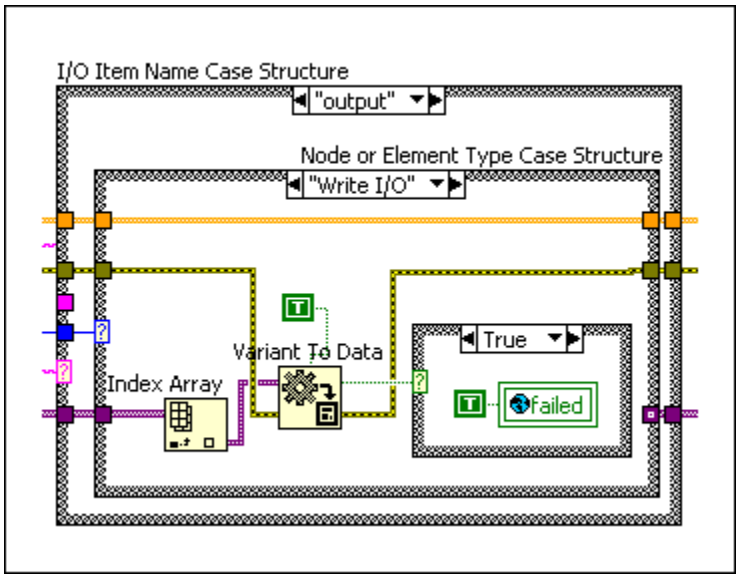
10. 运行Inverter VI。注意两个FPGA I/O节点间的连线的探针指示TRUE值，因为在用于反转VI的仿真I/O中包含了一个真常量。LabVIEW还将返回错误消息，指示用户未指定配置用于**输出**I/O项的FPGA I/O节点的行为。

测试输出I/O项

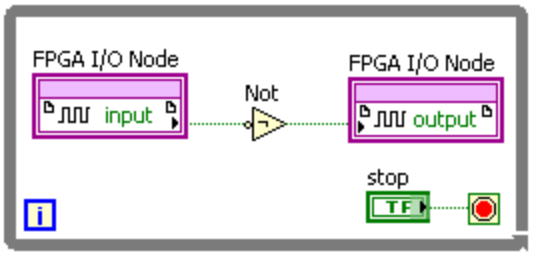
1. 在**项目浏览器**窗口中，选择**我的电脑**，然后选择**文件»新建**，显示**新建**对话

框。

2. 从**其他文件**文件夹中选择**全局变量**，新建一个全局VI。
3. 将全局VI另存为`Test Bookkeeping`。
4. 将一个名为**失败**的布尔显示控件添加至Test Bookkeeping VI的前面板。**失败**显示控件为一个全局变量。
5. 在Simulated IO for Inverter VI的程序框图上，右键单击**I/O项名称**条件结构的**输入**条件分支，然后从快捷菜单中选择**复制分支**。命名该条件分支为`output`。
6. 在**节点或元素类型**条件结构中，选择**读取I/O**条件分支，双击选择器并输入`Write I/O`。该条件分支用于**输出**I/O项的FPGA I/O节点。
7. 删除进入变体隧道的代码，添加“索引数组”函数、“变体至数据转换”函数和一个带有**失败**全局变量的条件结构，如下列程序框图所示。由于**输入**I/O项的值指定为TRUE，在Inverter VI中代码正确的情况下，Inverter VI写入**输出**I/O项的值应为FALSE。因此，如果Inverter VI将TRUE写入**输出**I/O项，测试则失败。



8. 运行Inverter VI并监控Test Bookkeeping VI。注意，**失败**显示控件的值为TRUE表明测试失败。
9. 停止运行Inverter VI。
10. 如要修复Inverter VI的问题，可在下列程序框图中添加一个“非”函数：

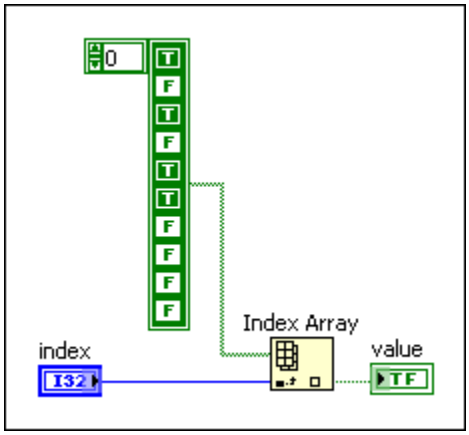


11. 在Test Bookkeeping VI中，右键单击**失败**显示控件，从快捷菜单中选择**数据操作»重新初始化为默认值**。当前如运行Inverter VI，请注意**失败**显示控件的值保持为FALSE，表明测试未失败。

使用更为复杂的自定义VI测试FPGA VI

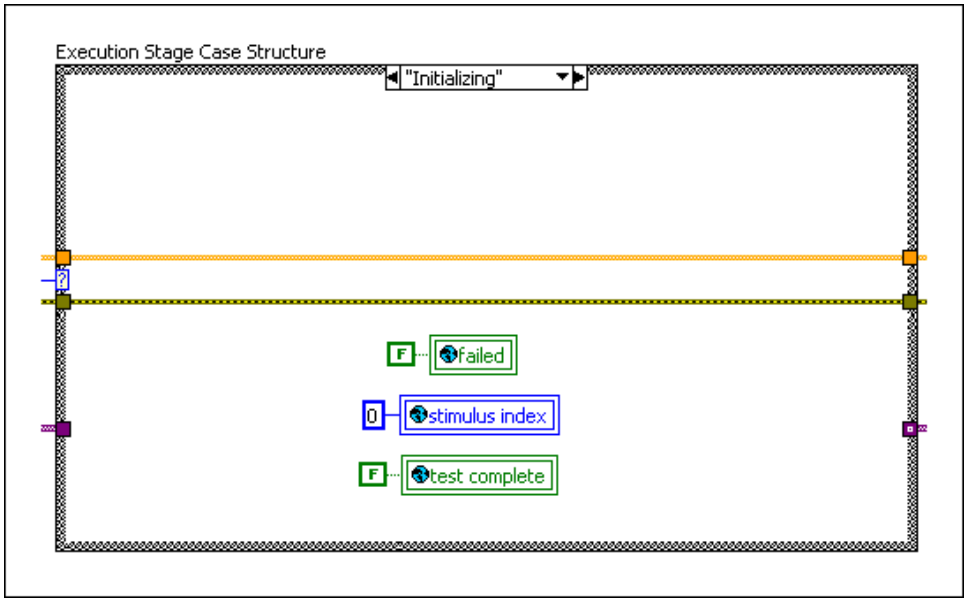
按照下列步骤测试Inverter VI对于10个选中的顺序输入值均能够正确操作。

1. 在**项目浏览器**窗口中，右键单击**我的电脑**，从快捷菜单中选择**新建»VI**。
2. 将该VI保存为`Test Stimulus.vi`。
3. 按照下列程序框图更新Test Stimulus VI：

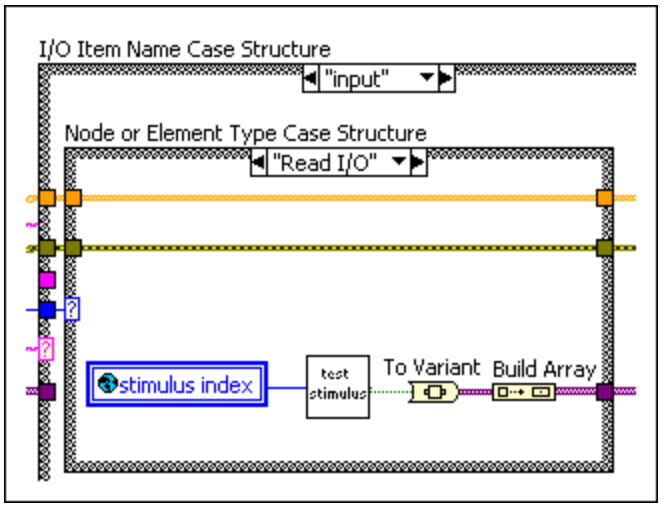


4. 为Test Stimulus VI创建连线板，以便在后续章节将其用作子VI。
5. 在Test Bookkeeping VI中，添加一个32位有符号整数的**激励索引**输入控件。同时添加一个**测试完成**布尔显示控件，如下列前面板中所示：

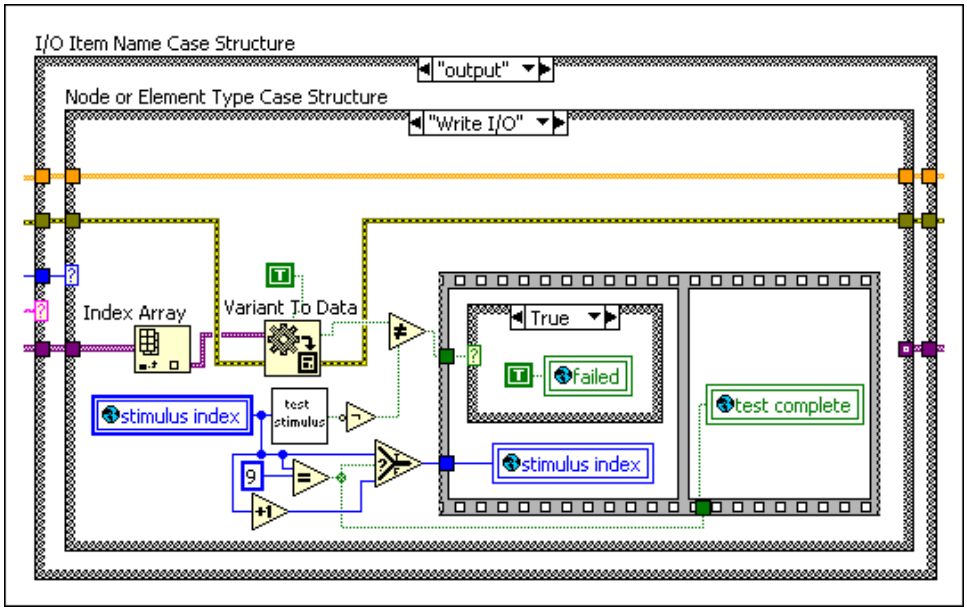
6. 在Inverter VI的仿真I/O中，更新**执行阶段**条件结构的**初始化**条件分支，以初始化来自Test Bookkeeping VI的值，如以下程序框图中所示。LabVIEW为**执行阶段**条件结构的每个条件分支创建一个自定义VI的副本。如要在不同的分支间共享数据可使用全局变量。如下列程序框图所示；或使用某些包含副本条件分支的机制。例如，非重入子VI。



7. 从**执行阶段**条件结构中选择**运行**条件分支，从**I/O项名称**条件结构中选择**输入**条件分支，从**节点或元素类型**条件结构中选择**读取I/O**条件分支。
8. 更新**节点或元素类型**条件结构的**读取I/O**条件分支的代码，如以下程序框图中所示：



9. 从**I/O项名称**条件结构中选择**输出**条件分支，从**节点或元素类型**条件结构中选择**写入I/O**条件分支。
10. 更新**节点或元素类型**条件结构的**写入I/O**条件分支的代码，如以下程序框图中所示。下列程序框图与输入的读数具有相同的激励、激励取反、比较FPGA I/O节点接收到的值的激励并在值不同时声明测试失败。程序框图还将递增**激励索引**，直至索引到达9。此时程序框图会将**测试完成**全局变量的值设为TRUE。

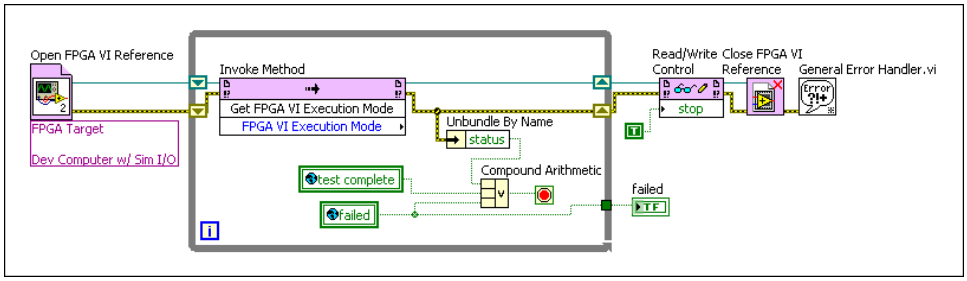


11. 运行Inverter VI并监控Test Bookkeeping VI中的全局变量。注意，**激励索引**递增至9，**失败**保持为FALSE，且**测试完成**保持为FALSE，直至**激励索引**的值到达9。
12. 停止运行Inverter VI。

使用主VI自动执行测试台

上述所创建的测试台需要手动执行某些操作，以运行中断结果。如要简化测试的重复性，请按照下列步骤创建一个主VI自动执行测试台操作。

1. 在**项目浏览器**窗口中，右键单击**我的电脑**，从快捷菜单中选择**新建»VI**。
2. 将VI另存为`Test Controller.vi`，并按照以下程序框图中所示更新VI：



Test Controller VI打开一个指向Inverter VI的引用并通过全局变量监控Inverter VI的执行。测试完成或生成错误后，将TRUE写入**停止**控件将停止Test Controller VI。配置用于“获取FPGA VI执行模式”方法的调用方法不会影响执行。但调用方法不提供访问发生在开发计算机上的FPGA VI执行过程中可能生成的错误的途径。

相关概念：

- 判定何时使用重入或非重入子VI

在FPGA终端上调试FPGA VI

如在FPGA终端上运行的应用中得到了未预期的数据，可使用下列方法识别并纠正FPGA VI或程序框图数据流的问题：

- 添加显示控件至FPGA VI程序框图，监控FPGA VI的内部状态。使用显示控件替代探头。在程序框图上需要查看数据的位置放置显示控件，以验证VI的功能。
- 添加I/O至FPGA VI程序框图。如在FPGA终端上有未用的I/O资源，可使用未用的资源监控布尔逻辑、触发器等的内部状态。

提示 某些情况下，在开发计算机上运行FPGA VI可缩短调试时间。

相关概念：

- 添加监控FPGA VI的显示控件
- 添加I/O至监控FPGA VI
- 使用仿真模式调试FPGA VI

添加I/O至监控FPGA VI

在FPGA VI中不能使用探头和其他传统的LabVIEW调试技术。如果FPGA终端上有未使用的I/O资源，则可以在FPGA VI程序框图中添加额外的I/O接线端进行调试。外部设备（例如，示波器、逻辑分析仪等）可用于监控额外的I/O接线端。使用额外的I/O接线端监控FPGA VI的范例如下：

- 在数字输出接线端上输出布尔逻辑的内部状态。
- 当指定部分的数据流执行期间，写入模式至多个数字输出接线端。
- 检测到触发时对数字输出接线端执行脉冲操作。
- 在循环的每次计数时触发数字输出接线端，以监控循环的性能。

例如，按照下列步骤在现有FPGA VI中的每个While循环的计数触发数字输出接线端，以监控While循环的性能。

1. 在**项目浏览器**窗口的同时包含FPGA I/O项的FPGA终端下新建一个VI或打开一个已有VI。
2. 新建一个FPGA I/O项或在**项目浏览器**窗口选择一个位于所用的FPGA终端下的未用FPGA I/O项。
3. 右键单击While循环，从快捷菜单中选择**添加移位寄存器**。
4. 在While循环内部添加一个FPGA I/O节点。
5. 单击FPGA I/O节点的**I/O名称**，选择之前添加至项目的FPGA I/O项。
6. 如FPGA I/O项为数字输入，右键单击FPGA I/O节点并选择**转换为写入**。
7. 连线左侧的移位寄存器至FPGA I/O节点的输入端。
8. 添加非函数至While循环。

9. 连线非函数的**x**输入端至移位寄存器和FPGA I/O节点。连线非函数的**非x?**输出端至右侧的移位寄存器。
10. 在**项目浏览器**窗口选择**文件»保存全部**保存对FPGA VI和LEP文件所作的改动。
11. 在FPGA终端上重新编译和运行FPGA VI。
12. 使用示波器或类似的设备监控选中的数字输出接线端的状态。

相关概念：

- 添加项至项目浏览器窗口中的FPGA终端
- 创建FPGA I/O项
- 编译、下载和运行FPGA VI
- 在FPGA终端上调试FPGA VI

添加监控FPGA VI的显示控件

在FPGA终端运行FPGA VI时，不能在FPGA VI中使用探针和其他传统的LabVIEW调试技术。但用户可在FPGA VI程序框图中使用显示控件。在程序框图上需要查看数据的位置放置显示控件，以验证VI的功能。待观察的信号比主机与终端的通信速度慢时，可使用此方法。

提示 使用控件更改FPGA VI的执行操作，可执行更高级的调试。

按照下列步骤使用显示控件，以替代现有FPGA VI中的探针。

1. 新建一个VI或打开一个FPGA终端下已包含FPGA I/O项的已有VI。
2. 右键单击待监视的连线，从快捷菜单中选择**创建»显示控件**。
3. 重复步骤2，在待监视的连线上放置显示控件。
4. 编译VI。

通过交互式前面板通信或可编程式FPGA接口通信运行FPGA VI，查看和验证FPGA VI中的数据。

相关概念：

- 添加项至项目浏览器窗口中的FPGA终端
- 编译、下载和运行FPGA VI
- 在FPGA终端上调试FPGA VI

使用第三方仿真器调试FPGA VI

随着FPGA设计体积和复杂程度的增加，用于编译和下载至终端的时间将越来越长。在FPGA芯片上的调试效率将会降低。配合其他调试技术，第三方仿真能够帮助您测试FPGA VI组件的定时动作。周期精确第三方仿真是指采用精确的定时，但仿真可不使用指定的硬件模型实现的仿真。

注： 并非全部终端均支持第三方仿真。如FPGA终端支持仿真，程序生成规范快捷键中将提供一个创建仿真导出的选项。

前提条件

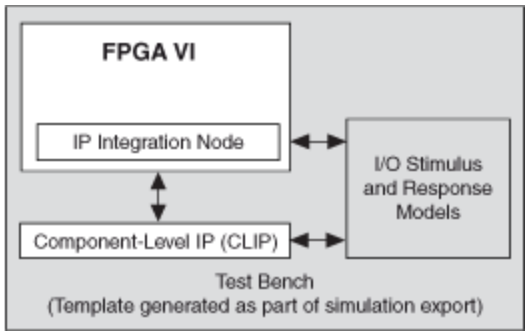
要使用第三方仿真，您必须熟悉Xilinx仿真器并安装硬件配置所需的Xilinx编译工具。

关于Xilinx编译工具支持的NI硬件的详细信息，见ni.com的技术支持文档。关于为LabVIEW安装Xilinx编译工具的说明，见Xilinx Compilation Tools Readme。

使用第三方仿真器模拟FPGA VI

如掌握VHDL语言，您可以使用Xilinx仿真器修改LabVIEW创建的VHDL测试台模板。需要为CLIP和IP集成节点中包含的IP提供仿真模型。通过各自的配置向导指定用于CLIP仿真和IP集成节点仿真的模型。

在下列示意图中，共同阴影区域表示组成测试台的部分。LabVIEW使用VHDL语言生成测试台模板。然后通过LabVIEW主控VI自定义与输入控件、显示控件和其他LabVIEW对象的交互。此外，可创建用于VHDL测试台内I/O的激励和响应模型。



相关概念：

- 使用DSP48E和DSP48E1函数的说明
- 调试FPGA VI
- 使用内置的控制逻辑实现FIFO
- 配置组件级IP向导

使用Xilinx仿真器调试FPGA VI

下文介绍了使用Xilinx仿真器和用户编辑的VHDL测试台模板调试FPGA VI的方法。

1. 配置LabVIEW，与Xilinx仿真器配合使用。
2. 如有需要可更改FPGA VI。例如，可能需要缩短仿真的运行时间。
3. 在**项目浏览器**窗口，右键单击FPGA终端，从快捷菜单中选择**选择执行模式»第三方仿真**。
4. 创建仿真导出程序生成范例。
5. 在**仿真导出属性**对话框中单击**生成**按钮，生成仿真导出。LabVIEW创建仿真所需的文件，并将其放在仿真目录下。
6. 通过修改模板中的VHDL代码，提供所需的激励和响应。
7. 在**项目浏览器**窗口，右键单击仿真导出并选择**打开仿真器**，以打开仿真器项目。
8. 单击**运行全部**按钮运行仿真。
9. 在波形查看器中查看信号并解决FPGA VI的故障。必要时可更改FPGA VI。
10. 必要的情况下集成更改至测试台。
11. 运行位于user目录下的RegenerateIsim.bat，重新生成仿真可执行程序。
12. 重复步骤6至11解决FPGA VI的故障。

配置用于第三方仿真器的LabVIEW

在导出FPGA VI进行仿真之前，必须先配置LabVIEW，使得LabVIEW与安装的仿真器可配合使用。

按照下列步骤，配置LabVIEW与安装的仿真器配合使用。

1. 在LabVIEW中，选择**工具»选项**，打开**选项**对话框。
2. 从**类别**列表中选择**FPGA模块**，显示**FPGA模块选项**对话框。
3. 从**仿真器**下拉菜单中选择一个仿真器。
4. 单击**确定**以保存修改。

缩短仿真运行时间

由于仿真软件的运行速度比FPGA硬件慢，因此并非全部FPGA VI均可仿真。用户可考虑限制FPGA VI的仿真部分，以缩短仿真运行时间。

下列策略可用于缩短仿真的运行时间：

- 减少节点（例如，离散延迟Express VI和零阶保持VI）的用户自定义延时。
- 减少FPGA定时函数的执行速率。

集成更改至VHDL测试台

在第三方仿真期间，您可以编辑FPGA VI并再次建立仿真输出。在某些情况下，您还必须将相应的改动集成到VHDL测试台中。

何时将改动集成到测试平台文件中

在以下情况下，您必须将更改集成到`user`目录下的测试平台文件：

- 编辑项目浏览器窗口中FPGA终端的属性。
- 更改项目浏览器窗口的项，例如，添加、移除或重命名FIFO。
- 在FPGA VI中添加或移除控件。

仿真输出的文件和目录结构

仿真输出为一组仿真所需的文件。必须访问仿真输出目录以编辑测试台文件。LabVIEW按以下目录结构组织仿真输出文件：

- 顶级目标目录—您在仿真导出属性对话框的信息页上指定的目录。
  - `user`—包含您必须编辑的测试平台文件。LabVIEW不会覆盖此目录中的文件。

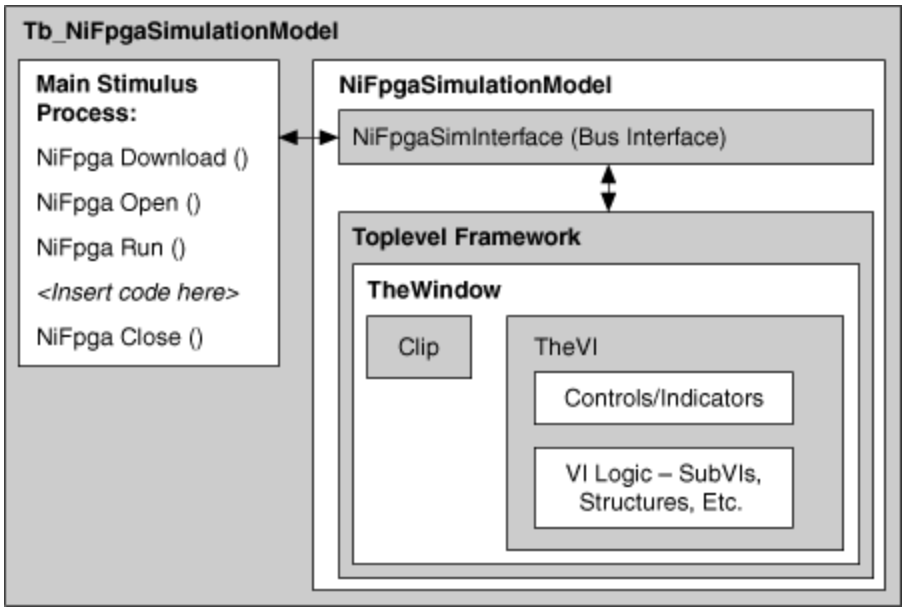
    > **注：** 如果安装的第三方仿真器是Xilinx仿真器，则必须运行`user`目录中的`RegenerateIsim.bat`，在更改生成的测试平台之后重新生成仿真可执行文件。

  - `niFpga`—包含最新的测试台模板和其他仿真文件。不要编辑该目录下的文件。每次生成仿真输出，LabVIEW将覆盖这个目录中的文件。
  - `isim`—（Xilinx仿真器）包含Xilinx仿真器的项目文件。您可以右键单击LabVIEW中的仿真导出生成规范，然后从快捷菜单中选择**启动仿真器**，在Xilinx仿真器中打开项目文件。

仿真VHDL架构

使用第三方仿真时，理解仿真输出VHDL架构有助于成功调试FPGA VI。

测试台结构`tb_NiFpgaSimulationModel`包含仿真所需的模型层次，如下图所示。



以下列表更详细地介绍了仿真测试台的组件。

tb_NiFpgaSimulationModel—总体测试台架构的默认名称。名称以`tb_`开始，名称的第二部分是在仿真导出属性对话框信息部分的**顶层仿真模型名称**。

- **主激励进程**—在这里添加代码以设置输入控件、读取显示控件，以及为其他FPGA VI执行其他激励。在`NiFPGA_Run`和`NiFpga_Close`之间输入想要执行的代码。
- **NiFpgaSimulationModel**—仿真模型。
  - **NiFpgaSimInterface**—总线接口，LabVIEW用来与终端特定代码通信的VHDL。名称以`NiFpgaSim`开始，并包括`interface`。名称的其他部分由终端决定。
  - **Toplevel架构**—特定终端的代码。模型的确切名称因终端而异。
    - **TheWindow**—要与之交互的代码。`TheWindow`将CLIP代码和FPGA VI代码进行分组。
      - **Clip**—在LabVIEW项目中指定的CLIP实例的CLIP仿真代码。您可以在配置组件级IP向导的名称和源页面上为CLIP指定仿真行为。该模型分组下的任意代码都是用户定义的。
      - **TheVI**—LabVIEW为FPGA VI生成的代码。LabVIEW分配给`TheVI`的名称来自仿真导出属性对话框源文件页指定的顶层VI的名称。
        - **输入控件和显示控件**—FPGA VI中输入控件和显示控件的代码。

该部分模型也包含处理FPGA VI重置的代码。

- **VI逻辑**—这部分模型包含从FPGA VI的程序框图运行VI逻辑的代码。该部分模型的代码层次结构反映了程序框图的层次结构。例如，模型中的单循环定时结构、平铺式顺序结构、条件结构包含VHDL代码，表示程序框图上的这些节点。

- （可选）I/O激励/响应—模型的这一部分包含仿真FPGA终端I/O所添加的代码。

TheVI下的VI逻辑

具有多个子程序框图的结构（条件结构、平铺式顺序结构等）的每个帧都有单独的VHDL实体对应。这些VHDL实体的名字以`_diag`以表明它们代表一个子程序框图。

可重入子VI的VHDL代码表现形式与结构相同。可重入子VI在VHDL代码层次结构中有多个实例。这些实例对应于程序框图中对子VI的调用。

不可重入的子VI包含特殊的VHDL逻辑来管理进入和来自重入子VI的数据流。

VHDL代码还包含表示LabVIEW VI和函数的实体。大多数这些元素都是加密的，但是您可以在其端口映射上看到传入和传出的值。

相关概念：

- 配置组件级IP向导

使用波形查看器查看信号

创建仿真导出时可指定待操作的信号类型，以在基于LabVIEW分组的波形查看器中查看上述信号。这些分组可帮助用户判定与LabVIEW程序框图组件相对应的信号。

注： 如要在波形查看器中使用信号分析必须熟悉VHDL编程语言及第三方仿真器。

波形信号器中的主要信号分组显示如下：

- 顶层输入控件和显示控件
- I/O
- CLIP顶层端口信号
- IP集成节点顶层端口信号

在主要组内，波形查看器同时也显示隐性信号。例如，时钟、启用链、寄存器更新逻辑、寄存器和程序框图连线。启用链强制执行LabVIEW数据流模式，因此可用其判定信号是否满足FPGA VI的整体执行顺序。仅可查看直接与程序框图信号关联的资源。

提示 使用程序框图对象的VHDL名称对应波形查看器中的信号对象。VHDL名称出现在即时帮助窗口的底部。

相关概念：

- FPGA VI的数据流和启用链

导出FPGA VI为Vivado Design Suite项目

LabVIEW FPGA模块提供了导出FPGA VI至Vivado Design Suite项目的选项。此选项允许用户设计导出的项目，并将其编译为Vivado Design Suite中的比特文件。您可以在FPGA模块中的FPGA目标上运行比特文件，例如Kintex-7 FlexRIO目标或高速串行设备。此选项利用了Vivado Design Suite提供的设计功能，同时充分利用了NI FPGA硬件资源。下文概述了导出过程。

注：

- 并非全部终端均支持Vivado Design Suite项目导出。关于终端的可用导出选项的详细信息，见指定FPGA终端的硬件文档。
- 必须在本地计算机上为Vivado安装必要的Xilinx编译工具，以将FPGA VI导出为Vivado Design Suite项目或打开导出的项目。关于用于Vivado不同版本的Xilinx编译工具所支持的NI硬件及LabVIEW版本，请

访问ni.com/info并输入信息代码XilinxCompileToolsZhs查询。有关为Vivado安装Xilinx编译工具的说明，见Xilinx Compilation Tools Readme。

导出FPGA VI

设计项目并将项目编译为可部署到FPGA终端的比特位文件之前，必须将FPGA VI导出为Vivado Design Suite项目。如要导出FPGA VI，首先必须创建程序生成规范。

注： 要允许在Vivado Design Suite中设计导出的项目，必须将第三方IP集成到FPGA VI中并命名设计文件，以便文件名以UserRTL_开头。例如，可定义VHDL文件名称为UserRTL_FpgaTop.vhd。除以UserRTL_开头的文件名外，所有为导出生成的设计文件均被加密。

按照下列步骤通过**项目浏览器**窗口导出FPGA VI。

1. 右键单击**项目浏览器**窗口的**程序生成规范**，从快捷菜单中单击**新建»Vivado的项目导出**显示Vivado Design Suite项目导出的属性对话框。或者右键单击Vivado Design Suite项目导出的现有程序生成规范，从快捷菜单中选择**属性**显示此对话框。
2. 在信息页指定程序生成规范的名称和其他描述性信息。
3. 打开源文件页指定顶层VI。FPGA VI仅可带有一个顶层VI。
4. 单击**确定**按钮关闭对话框或单击**生成**按钮导出FPGA VI。单击**生成**后，LabVIEW创建导出的必需文件并将其放置在用户在步骤2中指定的导出目录下。

注： 如在将FPGA VI导出为Vivado Design Suite项目后进行了进一步的更改，可在**程序生成规范**中右键单击Vivado Design Suite项目，并选择**生成**或**重新生成**将改动整合至现有的导出项目。

设计和编译导出的项目

导出FPGA VI为Vivado Design Suite项目后，可继续在Vivado Design Suite中设计和编译导出的项目。通过下列方式打开项目：

- 在**项目浏览器**窗口，右键单击**程序生成规范**下的项目并选择**打开Vivado Design Suite**。
- 在**项目浏览器**窗口，右键单击**程序生成规范**下的项目并选择**查看**打开导出目录。在导出目录中打开`LaunchVivadoDesignSuite.bat`。
- 浏览导出目录并打开`LaunchVivadoDesignSuite.bat`。

> **注：** 导出FPGA VI为Vivado Design Suite项目时，FPGA会自动在**项目设置**»**综合**中配置**-flatten_hierarchy**和**-keep_equivalent_registers**，并在Vivado Design Suite中的**Project Settings**»**Implementation**»**Write Bitstream (write_bitstream)**下配置**tcl.post**设置。**tcl.post**设置确保由导出项目生成的比特位文件能够被部署至NI FPGA终端。建议在Vivado Design Suite中设计项目时，保留上述设置。

关于设计导出项目并编译项目至比特位文件的详细信息，见Vivado Design Suite文档。

运行由导出项目生成的比特位文件

在Vivado Design Suite中为导出项目生成位文件后，必须在FPGA模块中创建主VI，以通过可编程FPGA接口通信编程下载或运行比特位文件。

参考资料

关于导出FPGA VI为Vivado Design Suite项目的详细信息，见设备指定范例。在LabVIEW中选择**帮助**»**查找范例**打开**NI范例查找器**，查看设备指定范例。单击**浏览**选项卡打开设备指定范例或单击**搜索**选项卡通过关键字搜索已安装的范例。

集成第三方IP

集成第三方IP至FPGA VI有2种方法：组件级IP(CLIP)接口和IP集成节点。本主题对两种方法进行比较，以帮助用户判定适合自身的FPGA应用。

支持的IP文件类型

可使用使用下列方法定义的第三方IP：

- VHDL
- Verilog
- 网表文件
- Xilinx IP配置文件：(Xilinx ISE) `.xco`文件或(Xilinx Vivado) `.xci`文件。

关于CLIP接口和IP集成节点支持文件类型的详细信息，见CLIP接口和IP集成节点详细信息章节。

注： 如要层次性导入Verilog文件或较大的文件，首先需要编译该IP至网表文件。

Xilinx IP

Xilinx提供并维护Xilinx IP。LabVIEW使用IP集成节点整合Xilinx IP至FPGA VI。由于Xilinx可能不再支持早期版本的IP内核。NI仅能确保支持下列IP集成节点，即该节点的Xilinx IP配置文件是通过用户FPGA终端上Xilinx编译工具的当前版本创建的。关于Xilinx编译工具支持的NI硬件的详细信息，见`ni.com`的技术支持文档。

注： Xilinx为Xilinx IP提供许可证。在指定Xilinx IP的**即时帮助**窗口，可查看许可证信息。如要导入许可证，根据所用的FPGA终端将`.lic`文件放在下列默认目录下：

- (Xilinx ISE) `C:\NIFPGA\programs\XilinxY_Z\ISE\coregen\core_licenses`，`XilinxY_Z`是FPGA终端的Xilinx编译工具（用于ISE）的当前版本。
- (Xilinx Vivado) `C:\NIFPGA\programs\VivadoA_B\data\ip\core_licenses`，`VivadoA_B`是FPGA终端的Xilinx编译工具（用于Vivado）的当前版本。

(Xilinx ISE)NI将Xilinx IP生成器(`coregen.exe`)安装在`LabVIEW.exe`同一目录下。默认情况下，Xilinx IP生成器位于`C:\NIFPGA\programs\XilinxY_Z\ISE\bin\nt`目录。`XilinxY_Z`是FPGA终端使用的Xilinx编译工具(ISE)的当前版本。

如将Xilinx IP从一个FPGA终端移植至另一终端，或从某一版本的LabVIEW移植至另一版本，可能需要在新的终端上重新生成IP。

CLIP接口和IP集成节点详细信息

下表详细地比较了两种集成第三方IP的方法。

注： 顶层VHDL文件仅支持`std_logic`和`std_logic_vector`端口类型。

相关概念：

- VHDL代码用作组件级IP
- 使用IP集成节点
- 集成Xilinx IP至FPGA VI
- 在FPGA VI中交互AXI IP

VHDL代码用作组件级IP

组件级IP(CLIP)用于实例化带有定义接口的VHDL代码，其占用部分FPGA资源。CLIP

可用于下列任务：

- 并行运行VHDL代码和LabVIEW代码。
- 在多个时钟域内执行VHDL代码。
- 编译包含限制条件。
- 创建CLIP时钟。
- 访问硬件I/O。（仅适用于部分终端。关于CLIP的I/O支持信息，见硬件文档。）

注： 如要使用CLIP，必须熟悉VHDL。

在FPGA应用中使用CLIP

下文为在FPGA应用中使用CLIP的详细步骤：

- 创建或获取IP。
- 使用配置组件级IP向导定义IP接口和创建声明XML文件。
- 使用“FPGA终端属性”对话框添加声明文件至项目。
- 添加CLIP项至项目。
- 在FPGA VI中使用CLIP项。

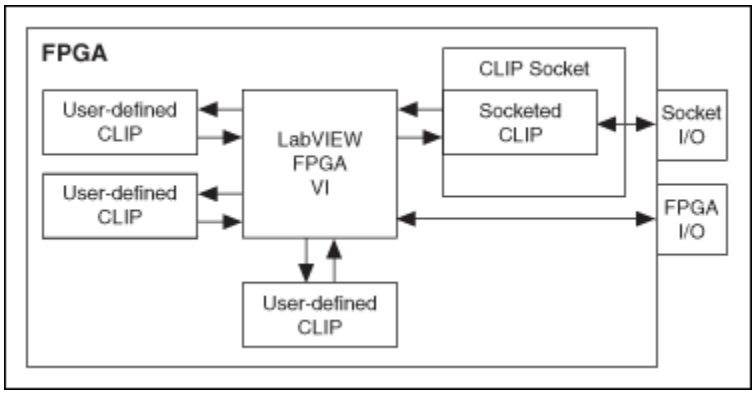
提示如果使用配置组件级IP向导创建或修改声明XML文件，LabVIEW将自动添加该文件至项目。

CLIP类型

是否支持CLIP取决于具体的FPGA终端。关于CLIP支持的信息，见终端硬件文档。部分FPGA终端支持下列一种或全部CLIP类型：

- **用户定义CLIP** – 允许VHDL代码直接与FPGA VI通信。
- **插槽CLIP** – 允许VHDL代码直接与FPGA VI通信，允许访问无法通过其他LabVIEW VI和函数访问的FPGA引脚。部分FPGA终端在FPGA中定义了固定的可插入插槽式CLIP的插槽。

下图显示了FPGA VI和CLIP之间的关系。



关于VHDL代码用作CLIP的范例，见CLIP入门指南：添加组件级IP至FPGA项目。

相关概念：

- CLIP入门指南－第一部分：创建VHDL代码
- 创建或采集IP
- 配置组件级IP向导
- 添加组件级IP至项目
- 在组件级IP和VI间传递数据
- CLIP入门指南：添加组件级IP至FPGA项目
- 集成第三方IP

配置组件级IP向导

通过配置组件级IP向导（CLIP向导）定义IP接口，而无需手动编辑声明XML文件。使用该向导创建或修改声明XML文件。CLIP向导还可用于检查CLIP使用的VHDL语法。根据要执行的操作，通过以下方式从FPGA终端属性对话框的组件级IP页打开CLIP向导。

- 单击**新建文件**按钮生成新的CLIP接口。
- 选择声明文件并单击**更改文件**按钮，修改现有的CLIP接口。

注： 如要使用该向导，用户本地计算机上必须安装所需的Xilinx编译工

> 具。关于Xilinx编译工具支持的NI硬件的详细信息，见ni.com的技术支持文档。关于为LabVIEW安装Xilinx编译工具的说明，见Xilinx Compilation Tools Readme。

CLIP向导包含下列页面：

1. 名称和源
2. 实体、架构、FPGA系列和IP类型
3. 类属
4. 基础信号配置
5. 附加时钟信号设置
6. 附加时钟状态信号设置
7. 附加数据信号设置
8. XML导出

配置用于仿真的CLIP

通过向导指定CLIP的仿真模型，以导出包含CLIP的FPGA VI，该CLIP用于第三方仿真。如要创建包含CLIP的FPGA VI的仿真导出，必须在向导的名称和源页面定义CLIP综合文件的仿真动作。

相关概念：

- 添加组件级IP至项目
- CLIP入门指南－第一部分：创建VHDL代码
- CLIP入门指南－第二部分：定义接口
- CLIP入门指南－第三部分：添加CLIP至项目
- 创建或采集IP
- 定义IP接口
- 仿真VHDL架构
- 使用第三方仿真器调试FPGA VI
- VHDL代码用作组件级IP

创建或采集IP

如要添加组件级IP (CLIP)至FPGA终端，必须以VHDL代码的形式提供IP，以将其编译至FPGA终端。提供VHDL代码的方式如下：

- 创建VHDL。
- 通过Xilinx IP生成器生成IP内核
- 通过Xilinx或合作伙伴购买IP。

支持的IP文件类型

可通过VHDL、Verilog或Xilinx IP生成器定义IP。使用通过Verilog定义的IP时，应先将IP编译为网表文件，然后在VHDL文件中封装该文件。使用通过Xilinx IP生成器定义的IP时，可直接导入约束文件至CLIP信号。根据指定的FPGA终端，使用约束文件指定约束。但如对CLIP信号应用约束，LabVIEW程序框图必须使用该信号。否则编译VI时，LabVIEW将返回Xilinx错误。

自定义用户库

在VHDL中，CLIP不支持自定义用户库。如VHDL使用了自定义用户库，可采用下列解决方案：

- 通过VHDL创建网表并使用CLIP集成该网表。
- 引用默认的引用库，而不是自定义用户库。

IP设计准则

为了满足FPGA终端要求以及正确匹配LabVIEW FPGA模块接口，VHDL必须满足几项准则。这些准则用于控制I/O端口、时钟及VHDL代码的重置操作。

I/O数据类型

所有与LabVIEW连接的I/O必须支持LabVIEW数据类型。下表列出了FPGA模块支持的数据类型以及必须用于IP接口的相应VHDL数据类型。

FPGA模块数据类型	VHDL数据类型
布尔	`std_logic`
U8和I8	`std_logic_vector(7 downto 0)`
U16和I16	`std_logic_vector(15 downto 0)`
U32和I32	`std_logic_vector(31 downto 0)`
U64和I64	`std_logic_vector(63 downto 0)`
定点	`std_logic_vector(x downto 0)`、其中x的范围为[0,63]。

例如，FPGA模块表示类型的IP输入端口`std_logic_vector(31 downto 0)`，即一个32位无符号整数(U32)。如VHDL代码需要使用一个不支持宽度（如`std_logic_vector(65 downto 0)`）的端口，必须创建一个封装VHDL文件，以将此端口扩展为FPGA模块支持的数据类型。关于该扩展范例的详细信息见NI网站。

注： 部分CLIP插槽支持额外的数据类型。CLIP插槽的可用性取决于FPGA终端。关于CLIP插槽详细信息见终端的硬件文档。

CLIP时钟

在XML文件中声明的`Reset`信号失效后，所有FPGA时基时钟和衍生时钟均稳定并立即开始自由运行。但外部时钟可能停止或产生毛刺。尽量避免使用可能停止或产生毛刺的时钟。如有重置时钟的代码，请确保其他代码的稳定和自由运行不会受到该时钟的影响。

访问CLIP I/O时必须考虑CLIP时钟。CLIP I/O定时是独立于IP的，因此IP必须能够使用正确的时钟读取和写入CLIP I/O。如在不同的时钟域读取和写入CLIP I/O，必须在FPGA I/O属性对话框的高级代码生成页面、FPGA I/O节点属性页面的高级代码生成页面和/或CLIP的VHDL代码内包含同步寄存器。但如果在同一时钟域内通过CLIP和VI访问同一I/O，设置同步寄存器的值为0可最小化延时。

也可在FPGA VI可用的CLIP内声明时钟。除CLIP时钟不能用作顶层时钟外，使用CLIP时钟与使用终端提供的FPGA时钟类似。

注： 插槽CLIP也可以直接访问FPGA的全局时钟。插槽CLIP的可用性取决于FPGA终端。关于插槽CLIP时钟的详细信息见终端的硬件文档。

重置VHDL代码

所有VHDL代码均包含一个信号异步重置输入，因此使用“重置”方法时，FPGA VI的VHDL代码被重置。建议包含信号异步重置。否则必须开发替代的重置机制，以确保VHDL代码在异步重置之前、之中和之后均正确运行。

约束和层次结构

编译时通过约束文件包含基于CLIP的用户约束，具体使用的格式取决于FPGA终端。该机制可用于除引脚布局约束外的全部约束。例如，可通过插槽CLIP的全局时钟缓存直接访问全局时钟输入引脚的时钟。必须约束该时钟的周期。

对于CLIP内指定组件的约束，可能需要指定组件在整个VHDL层次结构中的位置。对于这种情况可考虑使用下列宏作为限制的序。序能够保证约束的执行，而无需考虑组件在VHDL层次结构中的位置。如要使用此范例代码，可复制代码至文本文件，将文件另存为(Xilinx ISE) `DemoClipAdder.ucf`或(Xilinx Vivado) `DemoClipAdder.xdc`。在配置CLIP向导中添加该约束文件和VHD文件为综合文件，实现约束。

(Xilinx ISE)

```
NET "%ClipInstancePath%/myCLIPIO*" TNM_NET =
```

```
%ClipInstanceName%myCLIPIO; TIMESPEC
TS_%ClipInstanceName%ThruMyCLIP = TO
"%ClipInstanceName%myCLIPIO" 10 ns;
```

(Xilinx Vivado)

```
create_clock -period 10.000 -name %ClipInstanceName%Clk
-waveform {0.000 5.000} -add [get_pins
%ClipInstancePath%/clk] set_clock_latency -clock [get_clocks
{%ClipInstanceName%Clk}] 10.0 [get_pins
{%ClipInstancePath%/cAddOut[0]}]
```

如要多次实例化CLIP，每个CLIP实例必须具有唯一的名称，且名称需符合VHDL命名规范。唯一命名每个CLIP常量，且请勿将k用作CLIP常量名称的前缀。包含宏的情况下无需为每个实例包含独立的约束文件，因为FPGA模块创建了唯一的实例名称。

如未使用某个CLIP信号，Xilinx编译工具可能会从位流中移除该信号。此时可能在编译过程中产生NGBuild错误。如要解决该问题，可在FPGA VI中移除约束或使用该信号。

(Xilinx Vivado)对于每一次编译，FPGA模块都会生成一个文件名为unapplied_constraints.xdc的文件，列出所有未应用的约束条件。请按照下列步骤验证编译中是否应用了某个约束条件：

1. 找到默认目录：C:\NIFPGA。
2. 打开某次编译的文件夹。
3. 解压缩output_files.zip，将unapplied_constraints.xdc保存至易于访问的位置。
4. 打开unapplied_constraints.xdc，搜索约束条件。

关于创建VHDL代码的范例见“CLIP入门指南－第一部分：创建VHDL代码”。

警告 为确保数据完整性和时序收敛，验证来自CLIP的I/O节点的写入时钟域是否与其在LabVIEW程序框图上的读取时钟域相同，及验证至CLIP的I/O节点的读取时钟域是否与其

	在LabVIEW程序框图上的写入时钟域相同。

注： 请勿编写依赖于非CLIP逻辑路径的约束条件，因为NI无法保证这些路径不会更改。

相关概念：

- CLIP入门指南：添加组件级IP至FPGA项目
- CLIP入门指南－第一部分：创建VHDL代码
- VHDL代码用作组件级IP
- 配置组件级IP向导

定义IP接口

如要将IP添加为LabVIEW项目的组件级IP(CLIP)项，IP必须具备相关声明XML文件，以定义用于FPGA模块的I/O。声明XML文件用于描述IP的元素。FPGA模块使用该文件添加IP至LabVIEW项目。

有关使用XML的信息，请参阅`www.w3.org/XML`。

创建声明文件

注： NI强烈建议使用**配置组件级IP**向导创建此声明文件，这样可规避常见的错误。下表包含了**配置组件级IP**向导生成的XML标记的参考信息。

声明文件的第一行包含XML的版本信息。所有文件内部的标记都必须用`<CLIPDeclaration>`标记括起来，如下所示：

```
<?xml version="1.0"?><CLIPDeclaration Name="My VHDL IP">
<!-- Insert tags here --></CLIPDeclaration>
```

下表定义了可用于声明文件的XML标识符。

标签	必需?	父项标识符	带父标签的标签数量
CLIPDeclaration	是	—	—
FormatVersion	是	CLIPDeclaration	1
Description	否	CLIPDeclaration	1
SupportedDeviceFamilies	否	CLIPDeclaration	1
CompatibleCLIPSocketList	否	CLIPDeclaration	1
Socket	是	CompatibleCLIPSocketList	1 或更大
TopLevelEntityAndArchitecture	是	CLIPDeclaration	1

标签	必需?	父项标识符	带父标签的标签数量
SynthesisModel	是	TopLevelEntityAndArchitecture	1
Entity	是	SynthesisModel	1
Architecture	否	SynthesisModel	1
SimulationModel	否	TopLevelEntityAndArchitecture	1
Entity	是	SimulationModel	1
Architecture	否	SimulationModel	1
InterfaceList	是	CLIPDeclaration	1
Interface	是	InterfaceList	1 或更大
InterfaceType	是	Interface	1

标签	必需?	父项标识符	带父标签的标签数量
`SignalList`	是	Interface	1
`Signal`	是	SignalList	1 或更大

标签	必需?	父项标识符	带父标签的标签数量
Description	否	Signal	1
HDLName	否	Signal	0 或 1
RequiredClockDomain	否	Signal	0 或 1
LeaveUndrivenIfNotUsedInLabVIEW	否	Signal	0 或 1
HDLType	是	Signal	1

标签	必需?	父项标识符	带父标签的标签数量
UseInLabVIEWSingleCycleTimedLoop	否	Signal	0或1
SpecificTargetClock	否	Signal	0或1
DataType	是	Signal	1

标签	必需?	父项标识符	带父标签的标签数量
Boolean	否	DataType或Array	查看父标签
U8	否	DataType	查看父标签
U16	否	DataType	查看父标签
U32	否	DataType	查

标签	必需?	父项标识符	带父标签的标签数量
			看父标签
U64	否	DataType	查看父标签
I8	否	DataType	查看父标签
I16	否	DataType	查看父标签
I32	否	DataType	查看父标签
I64	否	DataType	查

标签	必需?	父项标识符	带父标签的标签数量
			看父标签
FXP	否	DataType	查看父标签
WordLength	否	FXP	0 或 1
IntegerWordLength	否	FXP	0 或 1
Unsigned	否	FXP	0 或 1
Array	否	DataType	查看父标签
Size	是	Array	1

标签	必需?	父项标识符	带父标签的标签数量
Direction	否	Signal	0或1
SignalType	否	Signal	0或1

标签	必需?	父项标识符	带父标签的标签数量
FreqInHertz	否	Signal	1
Max	是	FreqInHertz	1
Min	否	FreqInHertz	0 或 1

标签	必需?	父项标识符	带父标签的标签数量
`ImplementationList`	是	CLIPDeclaration	1
`Path`	是	ImplementationList	1 或更大

标签	必需?	父项标识符	带父标签的标签数量
MD5	否	Path	1
TopLevel	否	Path	0 或 1
SimulationFileList	否	Path	1
Path	否	SimulationFileList	1 或更大

标签	必需?	父项标识符	带父标签的标签数量
SimulationModelType	是	SimulationFileList	1
NumberOfDCMsNeeded	否	CLIPDeclaration	0或1

标签	必需?	父项标识符	带父标签的标签数量
NumberOfMMCMsNeeded	否	CLIPDeclaration	0 或 1
NumberOfBufGsNeeded	否	CLIPDeclaration	0 或 1
DutyCycleRange	否	Signal	0 或 1
PercentInHighMax	否	DutyCycleRange	0 或 1
PercentInHighMin	否	DutyCycleRange	0 或 1
AccuracyInPPM	在ToCLIP中：否 在FromCLIP中：是	Signal	0 或 1

标签	必需?	父项标识符	带父标签的标签数量
JitterInPicoSeconds	在ToCLIP中：否 在FromCLIP中：是	Signal	0或1
GenericList	否	CLIPDeclaration	1
Generic	是	GenericList	1或更大
Description	否	Generic	1
GenericType	是	Generic	1
DefaultValue	是	Generic	1
SupportDerivedClocks	否	Signal	1

标签	必需?	父项标识符	带父标签的标签数量
SourceClockReadyHDLName	否	Signal	1
DerivedClocksValidHDLName	是	Signal	1

标签	必需?	父项标识符	带父标签的标签数量
SupportsGating	ToCLIP：是 在 FromCLIP 中：否	Signal	1
CyclesRequiredBeforeAsynchronousResetClears	ToCLIP：是 在 FromCLIP 中：否	Signal	1

标签	必需?	父项标识符	带父标签的标签数量
ReadBeforePlace	否	Path	1

*该标签不包含值，因此您可以在打开标签中使用正斜杠或创建空的打开和关闭标签。例如，可以使用<Boolean />或<Boolean> </Boolean>来表示布尔数据类型。

**在CLIP参数中输入数值时，必须使用句点作为小数点分隔符。如使用逗号作为小

数点分隔符，LabVIEW将返回XML错误。

使用模式减少错误

XML模式文件是将XML文件限制为特定格式的定义文件。编写XML文件时，可以向大多数XML编辑工具添加模式文件。为用户定义CLIP创建CLIP声明文件时，可使用`labview\FPGA\CLIP\Schema`目录下的`CLIPDeclaration.xsd`模式文件最大限度地减少语法和格式错误。

注： `CLIPDeclaration.xsd`模式文件仅可用于用户定义CLIP。关于插槽CLIP模式的详细信息，见FPGA终端硬件文档。

注： NI强烈建议使用**配置组件级IP**向导，以避免声明XML文件中出现错误。

处理不同目录下的多个执行文件

用户可指定多个文件实现CLIP代码。在以下范例中，待执行的CLIP具有5个文件保存在下列目录结构中：

```
foo/myCLIPDeclaration.xml

foo/myTop.vhd

foo/mySide.vhd

foo/components/myLeftSubComponent.vhd

foo/components/myMiddleSubComponent.vhd

foo/components/myRightSubComponent.vhd
```

在`myCLIPDeclaration.xml`中可使用相对路径表明所有5个VHDL文件均为CLIP的必需文件。在XML文件中包含下列代码，指定`foo`目录下的两个VHDL文件和`components`中的所有文件均为必需文件。

```
<ImplementationList>
<Path Name="./myTop.vhd">
<TopLevel/></Path>
<Path Name="./mySide.vhd"></Path>
<Path Name="./components"></Path>
(
<Path Name="./components/myLeftSubComponent.vhd"></Path>
<Path Name="./components/myMiddleSubComponent.vhd"></Path>
<Path Name="./components/myRightSubComponent.vhd"></Path>
)
</ImplementationList>
```

请参考CLIP教程第2部分：定义CLIP接口的范例。

相关概念：

- CLIP入门指南：添加组件级IP至FPGA项目
- CLIP入门指南－第二部分：定义接口
- 配置组件级IP向导

添加组件级IP至项目

添加组件级IP(CLIP)项至LabVIEW项目，以实例化FPGA内部的CLIP。如要多次实例化CLIP，每个CLIP实例必须具有唯一的名称，且名称需符合VHDL命名规范。唯一命名每个CLIP常量，且请勿将k用作CLIP常量名称的前缀。

如要添加CLIP项至LabVIEW项目，必须关联CLIP声明文件至终端，然后通过声明文

件添加CLIP项至项目。通过“FPGA终端属性”对话框的“组件级IP”页面，关联CLIP声明文件至终端。通过“组件级IP属性”对话框的“常规”页面添加CLIP项至项目。通过**组件级IP属性**对话框的“时钟选择”页面链接FPGA终端的时钟至CLIP的输入端。

要在CLIP和FPGA VI间传递数据，CLIP声明文件必须具有`<InterfaceType>LabVIEW</InterfaceType>`标记。然后，添加CLIP项至LabVIEW项目时，FPGA模块将添加LabVIEW项目中的**CLIP**项下的声明XML文件定义的所有I/O。

提示 使用CLIP向导在声明文件中创建必要的接口。完成后，CLIP向导将自动添加CLIP至项目。

关于添加CLIP至项目的范例，请参见CLIP入门指南－第三部分：添加CLIP至项目。

相关概念：

- 在组件级IP和VI间传递数据
- 配置组件级IP向导
- CLIP入门指南－第三部分：添加CLIP至项目
- CLIP入门指南：添加组件级IP至FPGA项目
- VHDL代码用作组件级IP

在组件级IP和VI间传递数据

添加CLIP至LabVIEW项目时，FPGA模块添加CLIP I/O至LabVIEW项目。在FPGA VI中，可通过FPGA I/O节点读取或写入I/O。

默认情况下，CLIP信号使用同步寄存器。每个同步寄存器在VHDL代码接收到来自FPGA I/O节点的值前，增加一个时钟延时。如果CLIP已经在进入或来自FPGA VI的信号上包含触发器，可在LabVIEW中配置CLIP信号不使用同步寄存器。在FPGA I/O属性对话框的高级代码生成页面，设置**用于输出数据的同步寄存器数量**和**用于输出启用的同步寄存器数量**参数为0。如果CLIP与FPGA VI的时钟域相同，此时无需在CLIP中包含触发器或在LabVIEW中包含同步寄存器。

运行FPGA VI时，FPGA模块将FPGA VI和所有已初始化的CLIP编译至FPGA位流。

关于在CLIP和VI间传递数据的范例见“CLIP入门指南－第四部分：在CLIP和VI间传递数据”。

相关概念：

- 添加组件级IP至项目
- CLIP入门指南：添加组件级IP至FPGA项目
- CLIP入门指南－第四部分：在CLIP和VI间传递数据
- VHDL代码用作组件级IP

CLIP入门指南：添加组件级IP至FPGA项目

通过该入门指南可了解在FPGA项目中手动创建和使用CLIP的方法。但建议通过CLIP向导导入IP至FPGA项目为CLIP。

如要不通过CLIP向导添加CLIP至FPGA项目，必须按照下列步骤操作：

1. 创建或获取IP。
2. 使用声明XML文件定义至IP的接口。
3. 在FPGA终端属性中声明CLIP。
4. 添加CLIP项至LabVIEW项目。
5. 在CLIP和FPGA VI间传递数据。

下一节：CLIP入门指南－第一部分：创建VHDL代码

相关概念：

- 创建或采集IP
- 定义IP接口
- 添加组件级IP至项目
- 在组件级IP和VI间传递数据
- CLIP入门指南－第一部分：创建VHDL代码

- VHDL代码用作组件级IP

CLIP入门指南－第一部分：创建VHDL代码

如要添加组件级IP (CLIP)至FPGA终端，必须以VHDL代码的形式提供IP，以将其编译至FPGA终端。

入门指南使用带触发器的16位加法器。

下列VHDL代码获取2个I16数值的值，并以I16数值格式返回上述数值的和值。VHDL代码还需要来自FPGA模块的时钟和复位信号。如要使用此范例代码，可复制代码至文本文件，将文件另存为`DemoClipAdder.vhd`。

注： 为了保证应用所有约束条件，VHDL代码中包括`keep_hierarchy`属性，保证实例上的边界为静态，防止层次边界发生优化。

```
Library ieee; use ieee.std_logic_1164.all; use
ieee.numeric_std.all; entity DemoClipAdder is port ( clk :
in std_logic; aReset : in std_logic; cPortA : in
std_logic_vector(15 downto 0); cPortB : in
std_logic_vector(15 downto 0); cAddOut : out
std_logic_vector(15 downto 0) := (others => '0') ); end
DemoClipAdder; architecture rtl of DemoClipAdder is
attribute keep_hierarchy : string; attribute keep_hierarchy
of rtl : architecture is "yes"; begin process(aReset, clk)
begin if(aReset = '1') then cAddOut <= (others => '0');
elsif rising_edge(clk) then cAddOut <=
std_logic_vector(signed(cPortA) + signed(cPortB)); end if;
end process; end rtl;
```

为CLIP添加约束

下列UCF代码限制了和值输出的时钟速率不小于100 MHz。如要使用此范例代码，可复制代码至文本文件，将文件另存为(Xilinx ISE) `DemoClipAdder.ucf`或(Xilinx

Vivado) `DemoClipAdder.xdc`。在配置CLIP向导中添加该约束文件和VHD文件为综合文件，实现约束。

(Xilinx ISE)

```
NET "%ClipInstancePath%/cAddOut*" TNM_NET =
%ClipInstanceName%AddOut;TIMESPEC
TS_%ClipInstanceName%ThruAdder = TO
"%ClipInstanceName%AddOut" 10 ns;
```

(Xilinx Vivado)

```
create_clock -period 10.000 -name %ClipInstanceName%Clk
-waveform {0.000 5.000} -add [get_pins
%ClipInstancePath%/clk]set_clock_latency -clock [get_clocks
{%ClipInstanceName%Clk}] 10.0 [get_pins
{%ClipInstancePath%/cAddOut[0]}]
```


上一节：CLIP入门指南：添加组件级IP至FPGA项目	下一节：CLIP入门指南－第二部分：定义接口

相关概念：

- CLIP入门指南：添加组件级IP至FPGA项目
- VHDL代码用作组件级IP
- 创建或采集IP
- 配置组件级IP向导
- CLIP入门指南－第二部分：定义接口

CLIP入门指南－第二部分：定义接口

在LabVIEW项目中，如要添加IP为组件级IP(CLIP)项，IP必须带有用于定义I/O的声明XML文件，该I/O为用于LabVIEW的I/O。

使用配置组件级IP向导，通过范例DemoClipAdder.vhd文件自动创建该文件。按照

下列步骤，通过CLIP向导创建声明XML文件。

1. 在项目浏览器窗口，右键单击FPGA终端，从快捷菜单中选择**属性**。
2. 在FPGA终端属性对话框的组件级IP页面，单击**创建文件**按钮。
3. 在CLIP向导的名称和源页面，单击**添加综合文件**按钮。
4. 选择`DemoClipAdder.vhd`，单击**OK**按钮添加文件。
5. （可选）(Xilinx ISE)`DemoClipAdder.ucf`或(Xilinx Vivado)`DemoClipAdder.xdc`，单击**确定**按钮添加文件。
6. 单击**下一步**按钮。
7. 按照后续CLIP向导说明完成声明XML文件的创建。

   
   提示 单击向导每个页面的**帮助**按钮，查看可用选项的详细信息。

8. 单击**完成**按钮结束配置并创建声明XML文件。

CLIP向导自动添加CLIP至项目。

上一节：CLIP入门指南－第一部分：创建VHDL代码	下一节：CLIP入门指南－第三部分：添加CLIP至项目

相关概念：

- CLIP入门指南－第一部分：创建VHDL代码
- 定义IP接口
- 配置组件级IP向导
- CLIP入门指南－第三部分：添加CLIP至项目

CLIP入门指南－第三部分：添加CLIP至项目

添加组件级IP(CLIP)项至LabVIEW项目，以实例化FPGA内部的CLIP。

按照下列步骤添加CLIP项至项目：

1. 新建一个项目并将项目另存为`CLIP Demo.lvproj`。

2. 添加支持CLIP的FPGA终端至项目。
3. 右键单击FPGA终端，从快捷菜单中选择**属性**，显示FPGA终端属性对话框。
4. 从**类别**列表中选择**组件级IP**，显示组件级IP属性页面。
5. 单击**添加**按钮，选择`DemoClipAdder.xml`文件后单击**确定**按钮。

> **注：** CLIP向导自动添加声明XML文件至项目。

6. 单击**确定**按钮，关闭FPGA终端属性对话框。
7. 在项目浏览器窗口，右键单击FPGA终端，从快捷菜单中选择**新建»组件级IP**。
8. 在组件级IP属性对话框的常规页面，在**名称**栏输入加法器`CLIP`，在**组件级IP声明**下拉菜单中选择**DemoClipAdder**。
9. （可选）在时钟选择页面选择一个时钟。默认情况下，DemoClipAdder范例使用顶层的FPGA时钟。用户无需更改时钟。
10. 单击**OK**按钮。注意此时项目浏览器窗口已包含CLIP项，CLIP项与声明XML文件中的定义的I/O一致。

上一节：CLIP入门指南－第二部分：定义接口	下一节：CLIP入门指南－第四部分：在CLIP和VI间传递数据

相关概念：

- 添加组件级IP至项目
- CLIP入门指南－第二部分：定义接口
- 配置组件级IP向导
- CLIP入门指南－第四部分：在CLIP和VI间传递数据

CLIP入门指南－第四部分：在CLIP和VI间传递数据

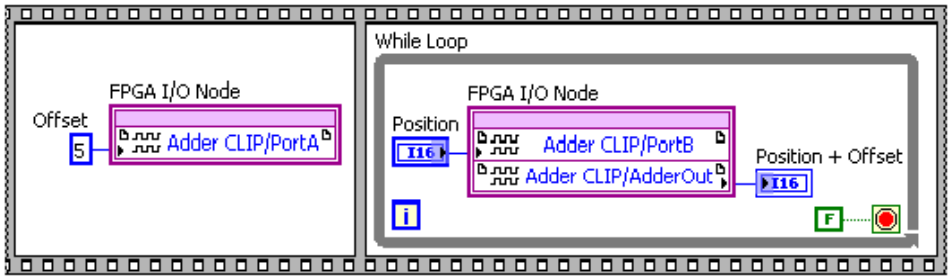
在FPGA VI中使用FPGA I/O节点在组件级IP(CLIP)和VI间传递数据。

按照下列步骤使用FPGA I/O节点访问CLIP I/O。

1. 在`Demo CLIP.lvproj`中，右键单击FPGA终端，从快捷菜单中选择**新建»VI**，

新建一个FPGA VI。

2. 添加平铺式顺序结构至FPGA VI。
3. 拖曳项目的**PortA** I/O项至“平铺式顺序结构”内部。
4. 右键单击**Adder CLIP/Port A**输入端，从快捷菜单中选择**创建»常量**，输入常量值5，命名常量标签为偏移量。
5. 右键单击“平铺式顺序结构”，从快捷菜单中选择**在后面添加帧**。
6. 拖曳项目的**PortB** I/O项至“平铺式顺序结构”的空白帧的内部。
7. 右键单击FPGA I/O节点的**Adder CLIP/Port B**输入端，从快捷菜单选择**创建»输入控件**，命名控件标签为位置。
8. 展开FPGA I/O节点，添加**AdderOut**。
9. 右键单击**Adder CLIP/AdderOut**输出端，从快捷菜单中选择**创建»显示控件**，命名控件标签为位置 + 偏移量。
10. 在While循环中包含FPGA I/O节点，如下列程序框图所示：



程序框图将常量（偏移量值）写入PortA输入端。在While循环中，FPGA I/O节点为位置添加偏移量，以创建新的位置值。

上一节：CLIP入门指南－第三部分：添加CLIP至项目

相关概念：

- CLIP入门指南－第三部分：添加CLIP至项目
- 在组件级IP和VI间传递数据

使用CLIP时钟

用户可在组建级IP(CLIP)中实现时钟电路，CLIP时钟的使用方法与终端提供的其它

时钟的使用方法类似。插槽式CLIP提供的外部时钟也可被映射至LabVIEW。CLIP时钟不能为顶层时钟。CLIP声明文件中的时钟将自动出现在项目浏览器窗口的CLIP项下。

当时钟不可用时，不能与CLIP时钟同时使用开始启用FPGA时钟和开始禁用FPGA时钟VI以保护电路。

在VHDL文件中包含时钟

如要确保CLIP时钟使用低斜率全局时钟网，必须使用全局时钟缓冲(BUFG)。NI建议使用带有门控输入的全局时钟缓冲(BUFGCE)，以确保时钟在具有毛刺或与时钟周期限制冲突时被禁用。关于在VHDL中创建时钟的信息见Xilinx documentation。

关于在VHDL代码中创建时钟的演示见用于CLIP时钟的VHDL代码的范例。

在CLIP声明文件中包含时钟

在CLIP声明文件中使用与定义I/O相同的语法定义时钟。时钟和I/O在声明文件中的出现顺序表明了其在LabVIEW项目中出现的顺序。对于CLIP时钟，必须定义下列标签。

- `JitterInPicoSeconds`
- `AccuracyInPPM`
- `DutyCyclePercentInMin/Max`

通过配置组件级IP向导（CLIP向导）定义IP接口，而无需手动编辑声明XML文件。

注： 如使用Xilinx数字时钟管理(DCM)电路衍生CLIP时钟，请使用FPGA终端的Xilinx specification sheets确保在CLIP声明文件中正确配置了CLIP时钟。下表给出了查找有关指定声明项的信息的具体位置。

CLIP声明标签	Xilinx文档的类型	查找信息时的建议搜索关键词
`JitterInPicoSeconds`	DC and Switching	Output Clock Jitter

CLIP声明标签	Xilinx文档的类型	查找信息时的建议搜索关键词
	Characteristics	
AccuracyInPPM	DC and Switching Characteristics	Output Clock Phase Alignment
DutyCyclePercentInMin/Max	用户指南	DCM attributes

添加CLIP时钟至LabVIEW项目

按照下列步骤添加CLIP时钟至LabVIEW项目：

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 添加CLIP项至项目。
4. （可选）在**项目浏览器**窗口的CLIP项下右键单击CLIP时钟，从快捷菜单中选择**属性**，打开FPGA CLIP时钟属性对话框。
5. （可选）在**名称**文本框中重命名时钟。对话框中的所有其它组件均显示为灰色，仅显示CLIP时钟的值。不能通过LabVIEW配置CLIP时钟。必须在CLIP声明文件中更新CLIP时钟，以更改时钟配置。
6. 单击**OK**按钮。

CLIP时钟的衍生时钟

通过CLIP时钟可生成衍生时钟。必须配置CLIP时钟在单个频率编译，以启用该选项创建新的衍生时钟。

按照下列步骤生成外部时钟的衍生时钟：

1. 按照下列说明添加CLIP时钟至LabVIEW项目：
2. 右键单击“项目浏览器”窗口中的CLIP时钟。从快捷菜单中选择**新建FPGA衍生时钟**显示FPGA衍生时钟属性对话框。
3. 配置时钟。
4. 在CLIP声明XML文件中定义下列标签：

    - SupportDerivedClocks
    - SourceClockReadyHDLName
    - DerivedClocksValidHDLName

在子VI中使用CLIP时钟

时钟常量或时钟控件可用于在子VI中引用CLIP时钟。

控件或常量的时钟名称必须与**项目浏览器**窗口中的时钟名称完全匹配。如名称不匹配，编译FPGA VI时将收到一条错误信息。使用FPGA时钟常量或控件的下拉菜单确保指定的时钟名称与项目中的时钟名称匹配。

使用非CLIP时钟访问CLIP I/O

在单周期定时循环中使用非CLIP时钟操作CLIP I/O时，必须考虑时钟域。如CLIP在一个时钟域内执行，FPGA VI使用不同的时钟域访问相应的CLIP I/O时，信号可在时钟域间错误的传输。此时，定时冲突分析窗口显示定时冲突。

如要避免在时钟域内错误的传输数据，可使用下列策略之一。

- 修改FPGA VI，使CLIP和单周期定时循环使用相同的时钟。
- 重新设计FPGA，以使用另一种方法执行多个时钟域
- 更改CLIP声明XML file，使CLIP I/O可与CLIP时钟使用同一个时钟域。

相关概念：

- CLIP时钟的范例VHDL代码
- 使用FPGA时钟和定时
- 添加FPGA终端至LabVIEW项目
- 编译、下载和运行FPGA VI

CLIP时钟的范例VHDL代码

在FPGA VI中可使用组件级IP(CLIP)时钟。下列代码为使用CLIP内部的Xilinx DCM生成衍生时钟及使用诸如相位偏移等功能的范例。代码还显示了锁定和重置DCM和将

BUFGCE用于可能停止的时钟的方法。

提示 如要使用此代码，可将代码复制至文本文件并另存为ClipGenerateClks.vhd。然后通过配置组件级IP向导，由ClipGenerateClks.vhd自动生成声明XML文件。

```
--------------------------------------------------------------------------------
--
-- File: ClipGenerateClks.vhd
-- Author: National Instruments Corporation
-- Date: April 2009
--
--------------------------------------------------------------------------------
--
-- Purpose:
--
-- This CLIP component generates clocks using a Xilinx DCM. A 40 Mhz clock is
-- used to derive an 80 Mhz clock, and a phase shifted version of 80Mhz clock
-- that is shifted by 180 degrees.
--
-- This example illustrates to use other Xilinx DCM features
-- such as phase shifting.
--
-- Signal naming convention:
--
-- All asynchronous signals are pre-fixed with a.
-- All Clk40 synchronous signals are pre-fixed with c40.
-- All Clk80FromDcm0 synchronous signals are pre-fixed with c80.
-- All metastable signals are suffixed with _ms.
--
-- Ports:
--
-- aDiagramReset : This is the LabVIEW FPGA asynchronous diagram reset.
--
-- Clk40 : This is the LabVIEW FPGA generated 40 Mhz clock. This
-- clock is stable and free-running only when aDiagramReset
-- is low.
--
-- ClkOut80 : This is an 80 Mhz clock generated from Xilinx DCM.
--
-- ClkOut80P180 : This is an 80 Mhz clock phase shifted by 180 degrees
-- generated by Xilinx DCM.
--
```

```
----------------------------------------------------------------------------

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity ClipGenerateClks is
  port (
    aDiagramReset : in std_logic;
    Clk40 : in std_logic;
    ClkOut80 : out std_logic;
    ClkOut80P180 : out std_logic
  );
end ClipGenerateClks;

architecture rtl of ClipGenerateClks is

  -- BUFG component declaration
  component BUFG
    port (
       I : in std_logic;
       O : out std_logic);
  end component;

  -- BUFGCE component declaration
  component BUFGCE
    port (
       I : in std_logic;
       CE : in std_logic;
       O : out std_logic);
  end component;

  -- DCM component declaration
  component DCM
    generic (
       CLKIN_PERIOD : real;
       CLK_FEEDBACK : string;
       CLKDV_DIVIDE : real;
       CLKFX_DIVIDE : integer;
       CLKFX_MULTIPLY : integer;
       CLKIN_DIVIDE_BY_2 : boolean;
       CLKOUT_PHASE_SHIFT : string;
       DESKEW_ADJUST : string;
       DFS_FREQUENCY_MODE : string;
       DLL_FREQUENCY_MODE : string;
       DSS_MODE : string;
```

```
      DUTY_CYCLE_CORRECTION : Boolean;
      PHASE_SHIFT : integer;
      STARTUP_WAIT : boolean
    );
    port (
      CLKIN, CLKFB, RST, DSSEN, PSINCDEC, PSEN, PSCLK : in std_logic;
      CLK0, CLK90, CLK180, CLK270,
      CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180 : out std_logic;
      LOCKED : out std_logic;
      STATUS : out std_logic_vector(7 downto 0);
      PSDONE : out std_logic
    );
  end component;

  constant kDcmResetDuration : positive := 3;
  signal c40ResetToDcm0_ms : std_logic := '1';
  signal c40ResetToDcm0 : std_logic_vector(kDcmResetDuration downto 1) := (others
=> '1');
  signal Clk0FromDcm0, ClkFbToDcm0 : std_logic;
  signal Clk80FromDcm0, Clk80P180FromDcm0, Clk80ThruBufg : std_logic;
  signal aLockedFromDcm0 : std_logic;
  signal c80Locked_ms, c80Locked : std_logic := '0';

begin

  -- Dcm0RstShiftReg -------------------------------------------------------
  -- This shift register makes sure that when the DCM reset asserts, it
  -- remains asserted for at least three clock cycles of DCM CLKIN. This
  -- is required for correct DCM locking sequence. Refer to the Xilinx FPGA
  -- user guide for more information.
  --
  -- The DCM should be reset when aDiagramReset asserts. This is important
  -- because some LabVIEW FPGA generated clocks (base or derived clocks) may
  -- not be valid when aDiagramReset is asserted. All LabVIEW FPGA clocks
  -- (in this case Clk40) are guaranteed to be stable and free-running
  -- immediately following the de-assertion of aDiagramReset.
  -- However, if externally generated clocks are used to source the DCM, other
  -- reset schemes would have to be used so that the DCM is kept in reset until
  -- the external clock is stable and free-running.
  ------------------------------------------------------------------------------
  Dcm0RstShiftReg : process (aDiagramReset, Clk40)
  begin
    if aDiagramReset = '1' then
      c40ResetToDcm0_ms <= '1';
      c40ResetToDcm0 <= (others => '1');
    elsif rising_edge(Clk40) then
```

```
      c40ResetToDcm0_ms <= '0';
      c40ResetToDcm0(1) <= c40ResetToDcm0_ms;
      for i in 1 to kDcmResetDuration-1 loop
          c40ResetToDcm0(i+1) <= c40ResetToDcm0(i);
      end loop;
  end if;
end process Dcm0RstShiftReg;
-- DCM0: ---------------------------------------------------------------------
-- This DCM generates a 80 Mhz and a phase shifted version of the 80 Mhz
-- clock shifted by 180 degrees.
-------------------------------------------------------------------------------
DCM0: DCM
  generic map(
      CLKIN_PERIOD => 25.0,
      CLK_FEEDBACK => "1X",
      CLKDV_DIVIDE => 2.0,
      CLKFX_DIVIDE => 1,
      CLKFX_MULTIPLY => 4,
      CLKIN_DIVIDE_BY_2 => FALSE,
      CLKOUT_PHASE_SHIFT => "NONE",
      DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
      DFS_FREQUENCY_MODE => "LOW",
      DLL_FREQUENCY_MODE => "LOW",
      DSS_MODE => "NONE",
      DUTY_CYCLE_CORRECTION => TRUE,
      PHASE_SHIFT => 0,
      STARTUP_WAIT => FALSE
  )
  port map (
      RST => c40ResetToDcm0(kDcmResetDuration),
      CLKIN => Clk40,
      CLKFB => ClkFbToDcm0,
      CLK0 =>Clk0FromDcm0,
      CLK180 =>OPEN,
      CLK270 =>OPEN,
      CLK2X =>Clk80FromDcm0,
      CLK2X180 =>Clk80P180FromDcm0,
      CLK90 =>OPEN,
      CLKDV =>OPEN,
      CLKFX =>OPEN,
      CLKFX180 =>OPEN,
      DSSEN =>'0',
      PSCLK =>'0',
      PSEN =>'0',
      PSINCDEC =>'0',
      LOCKED =>aLockedFromDcm0,
```

```
      PSDONE =>OPEN,
      STATUS =>OPEN);
-- BufgClkFbofDcm0: ------------------------------------------------------
-- This BUFG is used to drive the feedback clock input of DCM0.
-------------------------------------------------------------------------
BufgClkFbofDcm0: BUFG
  port map (
      I =>Clk0FromDcm0,
      O =>ClkFbToDcm0);
-- BufgClk80: ------------------------------------------------------------
-- This BUFG is used to drive the clock used by the synchronizer on locked
-- signal from DCM0.
-------------------------------------------------------------------------
BufgClk80: BUFG
  port map (
      I => Clk80FromDcm0,
      O => Clk80ThruBufg);
-- DblSyncLocked: --------------------------------------------------------
-- This double synchronizer synchronizes the asynchronous locked signal from
-- DCM0 to the Clk80FromDcm0 clock domain. This signal will be used to drive
-- the CE pin of the BUFGCE. This is because CE must not change during a short
-- setup window just prior to the rising clock edge on the BUFGCE input I.
-- Violating this setup time requirement can result in glitchy output pulse.
-- Refer to the Xilinx FPGA user guide for more information.
--
-- However, when aDiagramReset asserts, c80Locked could violate the setup
-- time requirement of Clk80FromDcm0, and it could result in a glitchy output
-- pulse at output O of BUFGCE. This glitch is acceptable because the FPGA VI
-- will be in an asynchronous reset. However, if this clock is being used for
-- other circuitry not reset by aDiagramReset, that circuitry should be
-- disabled before this reset asserts.
-------------------------------------------------------------------------
DblSyncLocked : process (aDiagramReset, Clk80ThruBufg)
begin
  if aDiagramReset = '1' then
      c80Locked_ms <= '0';
      c80Locked <= '0';
  elsif rising_edge(Clk80ThruBufg) then
      c80Locked_ms <= aLockedFromDcm0;
      c80Locked <= c80Locked_ms;
  end if;
end process DblSyncLocked;

-- BufgceClk80: ----------------------------------------------------------
-- This BUFGCE remains disabled until DCM0 achieves lock. This ensures that
-- the global clock network is driven only when the 80 Mhz derived clock is
```

```
  -- valid and glitch free.
  --------------------------------------------------------------------------------
  Bufgce_Clk80: BUFGCE
    port map (
       I =>Clk80FromDcm0,
       CE =>c80Locked,
       O =>ClkOut80);

  -- Bufgce_Clk80P180: ------------------------------------------------------------
  -- This BUFGCE remains disabled until DCM0 achieves lock. This ensures that
  -- the global clock network is driven only when the phase shifted 80 Mhz
  -- derived clock is valid and glitch free.
  --------------------------------------------------------------------------------
  Bufgce_Clk80P180: BUFGCE
    port map (
       I =>Clk80P180FromDcm0,
       CE =>c80Locked,
       O =>ClkOut80P180);

end rtl;
```

相关概念：

- 使用CLIP时钟

使用IP集成节点

使用IP集成节点将第三方IP集成至FPGA VI的程序框图。使用此节点包含下列任务。

1. 在打算用于配置IP集成节点的计算机上安装必需的Xilinx编译工具。
2. 创建或获取IP综合文件。例如，.vhd文件、Xilinx IP配置文件或网表文件。确保IP满足节点的要求。
3. 如考虑输出IP用于仿真，创建或获取定义仿真模型的文件。
4. 添加IP集成节点至程序框图。
5. 双击节点进行配置。LabVIEW将显示可完成下列任务的配置向导。
   a. 定义IP的名称及构成IP的综合文件，其中包括顶层综合文件。
   b. 设置每个综合文件的仿真动作。组合动作将生成节点的仿真模型。
   c. 如顶层综合文件为.vhd文件，定义顶层实体和架构。
   d. 指定IP可运行的FPGA系列。

e. 如顶层综合文件为 .vhd文件，设置任意类属的值并确保其语法有效。
f. 生成仿真模型。
g. 指定与时钟启用信号相关的IP端口。
h. 指定与同步或异步重置信号相关的IP端口。
i. 指定IP重置的时间和方式。
j. 定义出现的程序框图接线端并设置接线端的数据类型。

完成上述步骤后，LabVIEW生成集成IP至FPGA VI的代码。

IP集成节点属性向导

通过IP集成节点属性向导配置IP集成节点。

IP集成节点属性向导包含下列页面：

1. 名称和源
2. 实体、架构和终端
3. 类属和支持文件生成
4. 时钟和启用信号
5. 重置信号和行为
6. IP接线端

相关概念：

- 集成第三方IP

准备与IP集成节点配合使用的IP

下表给出了IP配合IP集成节点使用的建议和要求。

适用于所有IP的建议

下列建议适用于所有与IP集成节点配合使用的IP。

	建议	详细信息
❑	使用VHDL。	IP集成节点的设计宗旨是与VHDL语言配合使用，以发挥最佳性能。使用Verilog语言限制了可用的仿真选项。如要集成Verilog代码，请先编译代码为网表文件。然后可配合使用该文件和IP集成节点。
❑	验证LabVIEW外部的IP	IP集成节点不是调试或测试环境。集成IP至IP集成节点前，建议首先在Xilinx编译工具中综合验证IP。或者可在第三方仿真工具中创建一个测试平台，以确保IP的鲁棒性。
❑	同步IP输出端口至单周期定时循环时钟的上升沿。	同步输出端口至单周期定时循环时钟的上升沿可确保仿真FPGA VI可与VI在FPGA终端上运行产生相同的结果。

适用于所有IP的要求

下列要求适用于所有与IP集成节点配合使用的IP。

注： 顶层文件下层的端口可为任意数据类型。

包含顺序逻辑的IP的建议

顺序逻辑为使用一个或多个FPGA逻辑单元（例如，触发器）存储一个时钟周期至下一个时钟周期的状态的逻辑。检查IP以确认其是否包含顺序逻辑。包含的情况下，请应用下列建议至IP。IP不包含任意顺序逻辑的情况下，IP为组合逻辑。

注： 如IP不包含启用信号且FPGA VI重置，在单周期定时循环开始执行前，LabVIEW可能会丢弃IP的初始化值。缺失启用信号意味着无法阻止IP在无用数据的时钟周期内运行，而此时其它程序函数正在初始化。

仅包含组合逻辑的IP的建议

组合逻辑为不存储逻辑状态的逻辑，即非顺序逻辑。IP不包含任意顺序逻辑的情况下，IP为组合逻辑。如IP包含任意组合逻辑，请完成下列步骤：

1. 双击IP集成节点并打开时钟和启用信号页面。

> **注：** 如已配置节点，可右键单击节点，从快捷菜单中选择**配置»时钟和启用信号**。

2. 在**时钟信号名称**下拉菜单，选择**无时钟信号**。
3. 单击**确定**按钮保存更改，并返回程序框图。

IP集成节点将IP视作组合逻辑。

移动IP集成节点至另一台计算机

配置和使用IP集成节点，LabVIEW生成几个支持文件至磁盘。必须将上述文件与包含IP集成节点的VI放在一起。

如移动包含该节点的VI至磁盘上的另一文件夹、另一台计算机或提交VI至源代码控制软件，必须同时移动VI和这些文件。否则，用户需要在新的电脑上重新生成支持文件。

> **注：** 移动文件（包括源和支持文件）前，必须保持目录结构，其中包括包含IP集成节点的VI。

移动VI至另一台计算机时，请确保同时移动了下列文件：

- 在“名称”和“源”页面指定的综合文件包含下列生成的文件：
  - （如IP包含Xilinx IP配置文件）(Xilinx ISE) `xco_filename\XcoGeneratedFiles`目录或(Xilinx Vivado) `xci_filename\XciGeneratedFiles`目录下的所有文件。(Xilinx ISE) `xco_filename\XcoGeneratedFiles`目录与`.xco`文件本身位于同一目录下，或者(Xilinx Vivado) `xci_filename\XciGeneratedFiles`目录与`.xci`文件本身位于同一目录下。
  - （如IP包含`.ngc`文件）`ngc_filename\NgcGeneratedFiles\(Vivado/ISE)`目录下的

所有文件。ngc_filename\NgcGeneratedFiles\(Vivado/ISE)目录与.ngc文件本身位于同一目录下。

- （如IP包含.edif文件）edif_filename\EdifGeneratedFiles\(Vivado/ISE)目录下的所有文件。edif_filename\EdifGeneratedFiles\(Vivado/ISE)目录与.edif文件本身位于同一目录下。

- 在“设置仿真行为”对话框中指定的任意仿真文件。
- 所有生成的仿真文件，位于IP_name\SimFiles目录下。IP_name目录与顶层综合文件位于同一目录下。使用配置对话框“名称”和“源”页面的**IP名称**文本框定义IP名称。

> **注：** 请勿对一个以上的IP集成节点使用相同的**IP名称**。如对超过一个以上的IP集成节点使用相同的名称并生成仿真模型，LabVIEW将显示警告，以阻止用户覆盖另一个节点的仿真文件。

IP的执行速率高于包含该节点的单周期定时循环

可以使用IP集成节点集成IP，该IP的执行速度要高于包含该节点的单周期定时循环。在此情况下，IP必须使用FPGA衍生时钟，该时钟的执行速率为连线至单周期定时循环的时钟信号的整数倍数。例如，如单周期定时循环时钟的运行速率为100 MHz，用户可集成运行在100 MHz、200 MHz、300 MHz、400 MHz等以此类推的IP。

按照下列步骤配置IP集成节点，使其执行速度高于包含该节点的单周期定时循环。

1. 确保IP同步其输出端口至单周期定时循环时钟的上升沿。
2. 创建FPGA衍生时钟并将其设置为IP所需的速率。FPGA衍生时钟将显示为“项目浏览器”窗口中的项。

> **注：** FPGA衍生时钟必须与单周期定时循环时钟相位对齐。如不能实现相位对齐，“FPGA衍生时钟属性”对话框的**消息**栏将显示警告。当单周期定时循环时钟的频率位于相位对齐支持的频率范围外时，不能实现相位对齐。

3. 双击IP集成节点并打开时钟和启用信号页面。
4. 在**定时循环时钟衍生倍数**下拉菜单中，指定IP中与FPGA衍生时钟对应的端口。
5. 指定衍生时钟的**相对时钟频率**。例如，如时基时钟的执行速率为100 MHz，FPGA衍生时钟的执行速率为400 MHz，则输入`4x`。
6. 如FPGA衍生时钟带有启用信号，在**IP启用信号**列表中指定该信号。
7. 单击**下一个**按钮，待返回程序框图时单击**完成**按钮。
8. 添加FPGA时钟常量至程序框图并选择新建的FPGA衍生时钟。
9. 连线此常量的输出端至IP集成节点的**衍生时钟**输入端。

IP集成节点当前的执行速率要高于包含该节点的SCTL。

综合文件和仿真

添加综合文件至IP集成节点后，LabVIEW将为该文件分配一个仿真行为。用户可选择接受该行为、更改行为或从仿真中排除该文件。下表介绍了可用的综合文件类型、LabVIEW自动分配给每个文件类型的仿真行为以及导出用于Virtex-II FPGA仿真的特定文件类型的说明。

注： 即使不打算导出用于仿真的FPGA VI，也必须定义每个综合文件的仿真行为。

综合文件类型	LabVIEW分配的仿真行为	说明	Virtex-II FPGA的仿真说明
网表文件	**用户定义**	LabVIEW使用用户定义的仿真模型。必须单击**设置仿真行为**按钮打开设置仿真行为对话框。在该对话框中，使用**添加文件**、**删除文件**和**设置为顶层**按钮添加`.vhd`文件至仿真模型。通过提供网表文件的源可获取上述文件。确保网表文件名称与网表组件名称相同。如上述名称不同，IP集成节点属性向导的类属和支持文件生成页面将显示Xilinx错误和IP集成节点配置错误。	如IP包含专用于Virtex-II FPGA的组件（例如，`BCSAN`组件），则不能导出用于仿真的IP。

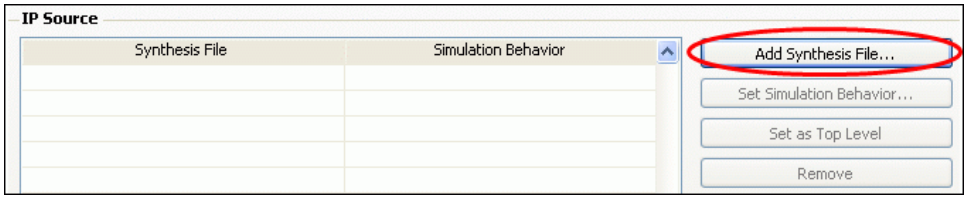
综合文件类型	LabVIEW分配的仿真行为	说明	Virtex-II FPGA的仿真说明
Xilinx IP配置文件	**与综合相同**	LabVIEW使用Xilinx IP生成器生成的.vhd仿真模型。	不能导出用于Virtex-II FPGA仿真的Xilinx IP配置文件。
.coe	**与综合相同**	N/A。	N/A。
.vhd和其他文件	**与综合相同**	LabVIEW使用.vhd综合文件仿真。	仅限.vhd文件：如IP实例化专用于Virtex-II FPGA的组件，NI不能确保仿真精度。

添加综合文件

综合文件包含要集成至FPGA VI的HDL代码。按照下列步骤添加综合文件至IP集成节点。

注： LabVIEW假定用户添加的一个文件为顶层文件。

1. 添加IP集成节点至程序框图。
2. 双击节点进行配置。此时将出现**名称和源**页面。
3. 单击**添加综合文件**按钮：



LabVIEW将提示您添加一个综合文件。

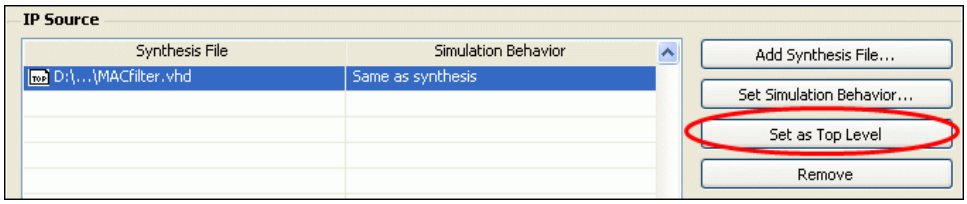
4. 选中要添加的综合文件并单击**确定**按钮。LabVIEW将显示**IP源**表中**综合文件**栏中的综合文件。
5. （可选）设置该文件为顶层文件。
6. 注意该表中的**仿真行为**栏。LabVIEW根据添加的文件类型分配仿真行为。可选择以下选项：
   - 接受LabVIEW分配的仿真行为。
   - 更改仿真行为。
   - 从仿真中排除此文件。

定义顶层综合文件

顶层综合文件定义IP集成节点综合的实体。如顶层文件为 .vhd文件，该文件同时还定义所用的架构。此外，顶层综合文件包含一系列对应于节点输入和输出接线端的端口。

LabVIEW假设用户添加的第一个 .vhd文件、Xilinx IP配置文件或网表文件为顶层文件。按照下列步骤指定另一个用作顶层综合文件的文件。

1. 确保文件符合顶层综合文件的要求。
2. 如尚未进行上述操作，可添加综合文件至IP集成节点使用的综合文件列表。
3. 在**IP源**表中选择文件。
4. 单击**设置为顶层**按钮：

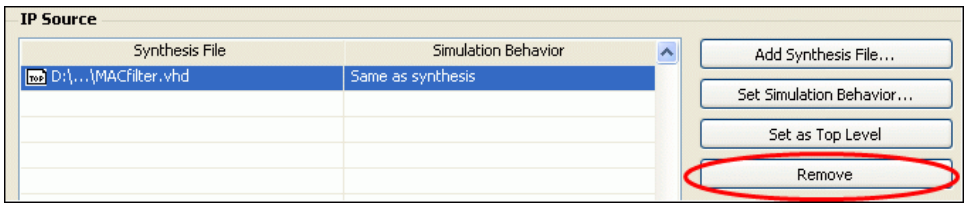


LabVIEW在顶层文件的左侧显示一个按钮。

删除综合文件

按照下列步骤，从**IP源**表中删除综合文件。

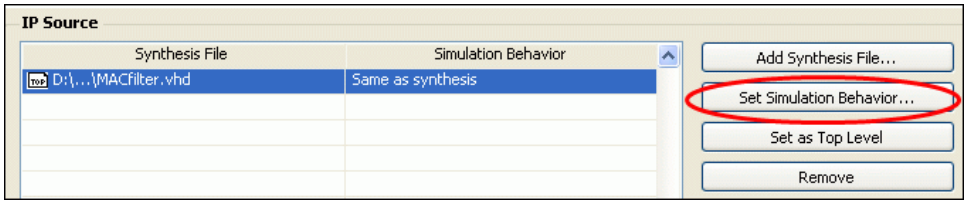
1. 在**IP源**列表中选择文件。
2. 单击**删除**按钮：



修改仿真动作

按照下列步骤更改LabVIEW分配给综合文件的仿真行为。

1. 在**IP源**表中选择一个或多个综合文件。
2. 单击**设置仿真行为**按钮：



LabVIEW将显示设置仿真行为对话框。

3. 选择一个新的仿真动作并正确配置。需要时可从仿真模型中排除此文件。

注：

- LabVIEW根据选中的综合文件类型，以灰色显示特定的仿真行为。
- 建议为网表综合文件选择**用户定义**选项。该选项需要一个或多个定义仿真模型的`.vhd`文件。如不具有上述`.vhd`文件，可选择**后综合模型**选项，使LabVIEW在仿真中生成一个`.vhd`文件。但由于次优化的VHDL生成，此选项将大幅增加仿真的运行时间。

- 不能为网表综合文件选择**与综合相同**选项。

4. 单击**确定**按钮返回配置向导。

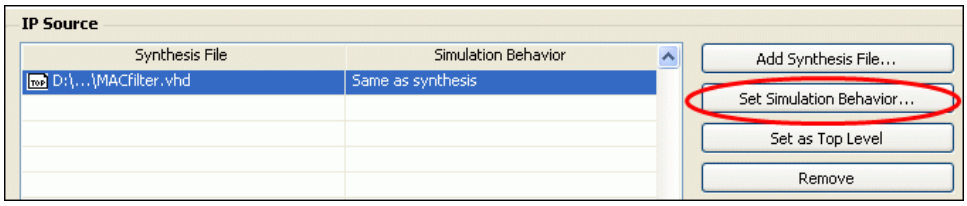
从仿真中排除一个综合文件

如不需要仿真一个综合文件，或另一个仿真模型中已包含了综合文件的仿真模型，此时可从仿真中排除此文件。排除文件将降低仿真过程中生成错误的几率。

注： 即使不打算导出用于仿真的FPGA VI，不能从仿真中排除顶层综合文件。

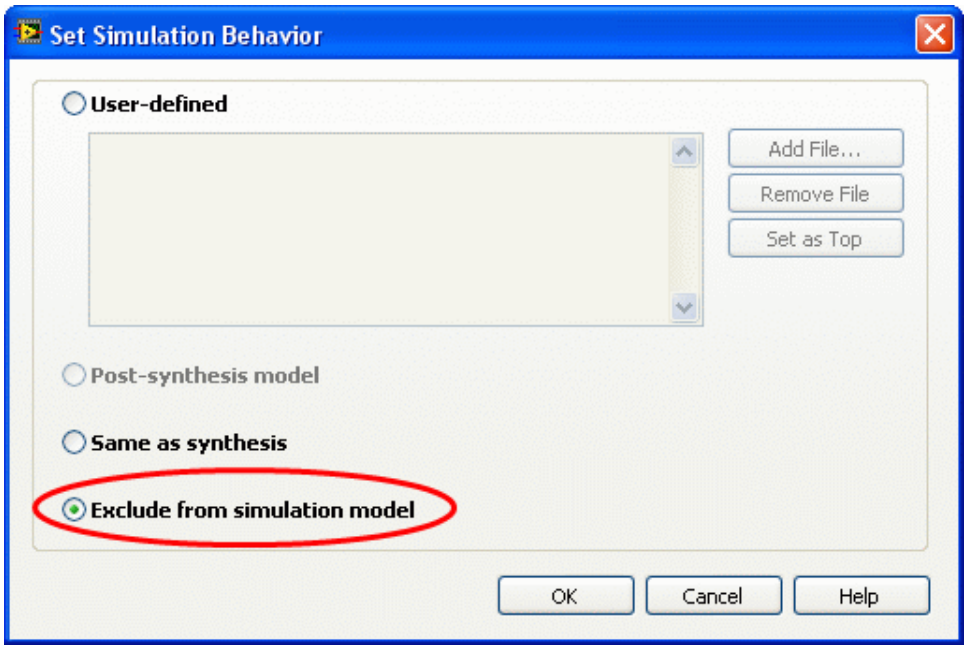
按照下列步骤从仿真中排除综合文件。

1. 在**IP源**表中选择一个或多个综合文件。
2. 单击**设置仿真行为**按钮：



LabVIEW将显示设置仿真行为对话框。

3. 选择**从仿真模型中排除**选项：



4. 单击**确定**按钮返回配置向导。

生成用于IP集成节点的支持文件

即使不打算导出用于仿真的FPGA VI，最终完成配置和使用IP集成节点前也必须生成支持文件。按照下列步骤生成支持文件。

1. 在IP集成节点配置向导的“名称和源”页面，确保每个综合文件具有所需的仿真行为。
2. 浏览“类属和支持文件生成”页面。
3. （可选）如顶层综合文件为包含VHDL类属的.vhd文件，根据具体情况设置这些类属的值。使用**检查语法**按钮确保输入的语法有效。
4. （可选）如要重新生成全部仿真模型，取消勾选**仅重新生成过期支持文件**复选框。否则，LabVIEW仅生成自上次生成仿真模型发生改动的综合组件。
5. 单击**生成**按钮生成支持文件。

仿真配置范例

下文提供了添加综合文件和配置仿真行为的范例。

同时用于仿真的.vhd综合文件

按照下列步骤添加.vhd文件，并将该文件用于仿真。

1. 添加IP集成节点至程序框图。
2. 双击节点打开节点配置对话框。
3. 在第一个页面上单击**添加综合文件**按钮。
4. 选择.vhd文件并单击**确定**按钮。该文件将出现在表格的**综合文件**一列。注意，**仿真行为**为**与综合文件一致**，即LabVIEW也将该.vhd综合文件用于仿真。

当前可继续配置IP集成节点。

带有几个.vhd文件的网表综合文件（用于仿真）

按照下列步骤添加网表文件，然后使用多个.vhd文件进行仿真。

1. 添加IP集成节点至程序框图。
2. 双击节点打开节点配置对话框。
3. 在第一个页面上单击**添加综合文件**按钮。
4. 选择网表文件并单击**确定**按钮。该文件将出现在表格的**综合文件**一列。
5. 单击**设置仿真行为**按钮打开“设置仿真行为”对话框。
6. 选择**用户定义**选项。
7. 使用文件列表右侧的按钮添加定义仿真的.vhd文件。确保正确设置顶层仿真文件。

当前可继续配置IP集成节点。

集成Xilinx IP至FPGA VI

LabVIEW使用IP集成节点整合Xilinx IP至FPGA VI。按照下列步骤添加Xilinx IP至FPGA VI。

1. 在支持的FPGA终端下新建一个空白VI，并显示VI的程序框图。
2. 右键单击程序框图，**选择编程**»**Xilinx IP**选板。

> **注：** 该选板仅显示FPGA设备系列支持的IP。并非全部FPGA设备系列均支持所有IP。关于FPGA系列支持的详细信息，见IP的数据表。

3. 选择所需的IP并将其放置在程序框图上。LabVIEW创建一个表示该IP的节点。
4. 双击节点打开节点配置对话框。
   a. 键入**IP名称**。LabVIEW将在程序框图图标上显示该名称。
   b. 指定**用于支持文件的文件夹**。该文件夹为Xilinx IP生成器放置必需文件的路径。如将该FPGA VI移至其他计算机，必须同时移动该文件夹。
   c. 单击**配置Xilinx IP**打开Xilinx IP生成器。
   d. 按照需要配置IP。使用**< 返回**和**下一步 >**按钮查看不同的选项。根据指定的FPGA终端，单击**Datasheet**按钮或**Documentation** 选项，查看包含IP详细信息的PDF文档。
   e. 配置IP后，根据指定的FPGA终端，单击**Generate**或**OK**。Xilinx IP生成器开始生成VHDL代码并返回LabVIEW。VHDL进程完成后，该进度条显示生成IP成功。
5. 单击**下一步**执行配置页面的其他操作。单击**完成**按钮结束配置Xilinx IP。

相关概念：

- 集成第三方IP

Xilinx IP列表

使用Xilinx IP函数，在FPGA VI中实现不同Xilinx IP。有关FPGA模块中可用的Xilinx IP函数的列表，请参考Xilinx IP选板。Xilinx IP名称来自Xilinx IP数据表，可登陆Xilinx网站www.xilinx.com查看。关于Xilinx IP的详细信息，请参考Xilinx IP的数据表。

> **注：** Xilinx提供并维护Xilinx IP。因为Xilinx可能弃用或更新Xilinx IP，NI仅支持当前版本Xilinx编译工具创建的Xilinx IP配置文件，这些配置文件将用于FPGA终端。关于每种Xilinx编译工具支持的NI硬件的详细信息，请访问 `ni.com/info`并输入信息代码XilinxCompileToolsZhs查询。关于弃用的Xilinx IP列表，请参考Xilinx网站。

相关概念：

- 在FPGA VI中交互AXI IP

在FPGA VI中交互AXI IP

NI为位于指定硬件终端上的特定Xilinx IP开放AXI（Advanced eXtensible Interface，高级可扩展接口）协议。AXI协议是用于互连用于高性能和高频应用程序的IP函数块的标准工业级总线接口。AXI协议共有三种类型：AXI4、AXI4-Lite和AXI4-Stream。

> **注：** LabVIEW FPGA模块的AXI协议支持随终端变化。Xilinx IP函数选板仅显示FPGA设备终端支持的IP。关于接口和FPGA设备支持的详细信息，见IP数据表。

FPGA VI中的AXI和4线IP接口

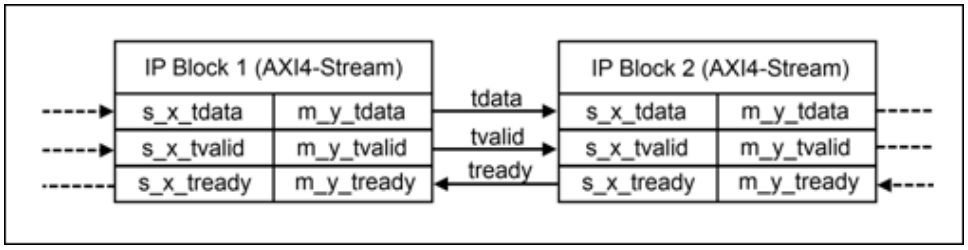
AXI协议类似于信号连接的LabVIEW 4线握手协议。AXI协议与LabVIEW 4线握手协议的主要区别是：在单周期定时循环中互连IP时的信号命名和反馈节点的放置。

Xilinx IP中的AXI信号的输入/输出接线端遵循常规命名规范： m/s + 协议 + 名称 + tdata/tvalid/tready。其中m为主控块或生成值的节点，s表示从控块或消费值的节点。例如，信号名称m_axis_signal_tvalid。其中m表示主控、axis表示AXI Stream协议、signal表示信号名称及tvalid表示有效。

通过NI IP创建的AXI信号的命名规范与Xilinx IP类似，但省略了axis关键词：m/s + 信号名称 + tdata/tvalid/tready。

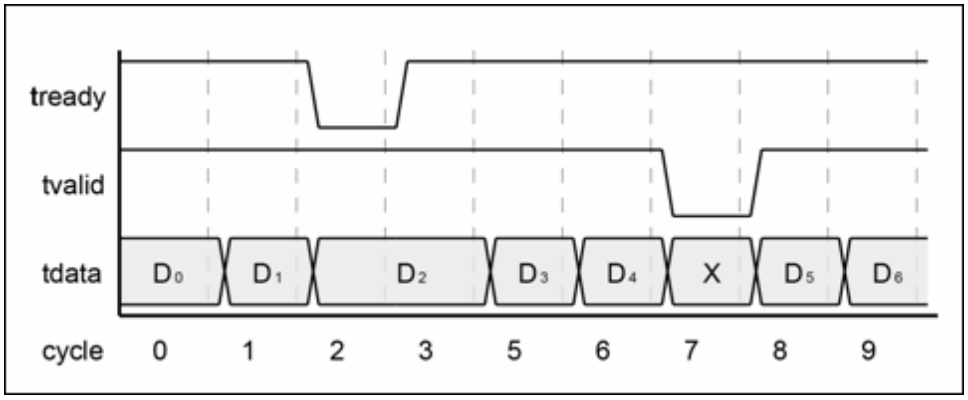
AXI至AXI互连

下列程序框图为两个AXI4-Stream IP块之间互连的概念性配置。



流向下方数据流的信号与4线握手协议中的信号功能一致。它们通知下方数据流块关于来自上方数据流块数据的有效性信息。例如，假设两个简单块的输入信号为**x**，输出信号为**y**。只要**m_y_tdata**输出包含有效的**y**数据，IP块1的**m_y_tvalid**输出为TRUE。

LabVIEW 4线握手协议和AXI Stream协议的主要区别在于反馈信号的含义，其通过流向处理链上方的数据线表示。AXI信号通知上方数据流块当前周期的读取状态，4线信号通知上方数据流块下一个周期的读取状态。采用AXI接口，块将**tready**信号置为无效，以通知上方数据流的计数部分，当前时钟周期不能接收数据。下图的定时框图的最初几个时钟周期描述了上述场景。注意，**tready**信号被重置为有效前，**tdata**的值保持。

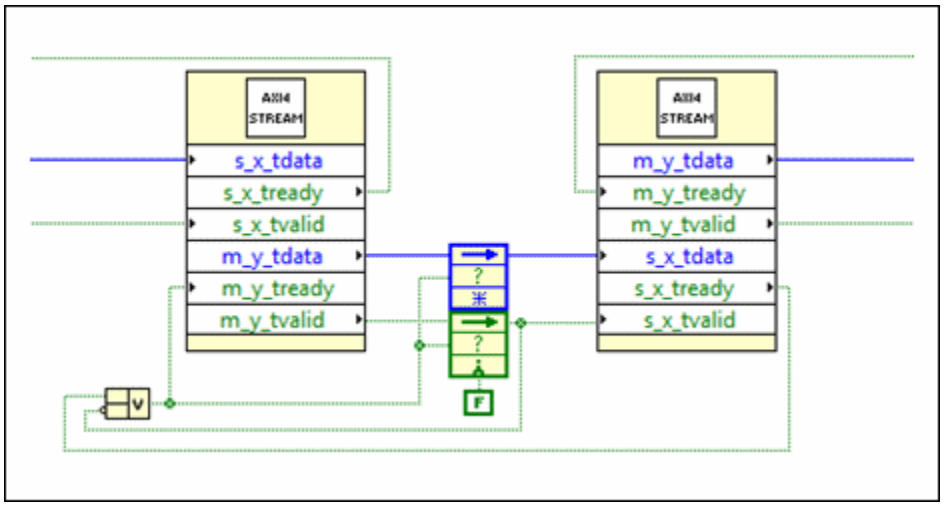


上图的定时框图中的后续时钟周期内，主控将**tvalid**信号置为无效，中止传输。

在LabVIEW的单周期定时循环内与AXI IP交互时，IP块 1 (AXI)的输出信号通过反馈节点连接至下方数据流IP块 2 (AXI)。反馈节点与之前路径中的寄存器功能一致。**s_x_tready**信号置为无效时，反馈节点将临时存储数据，直至从控块准备就绪接收新数据。

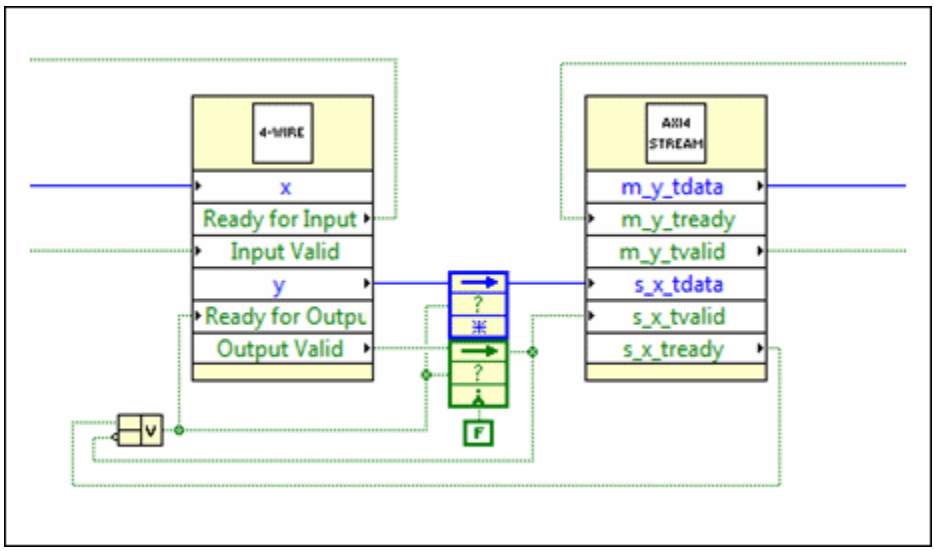
采用AXI4-Stream协议，从控块准备就绪生成有效数据前，主控块不允许等待。实

现遵从AXI协议的块时，只要关联的tready信号保持置为无效，该块必须重复生成相同的输出（有效值）。此外，需要使用“或(OR)”门控触发握手进程。如下列程序框图所示。



4线至AXI互连

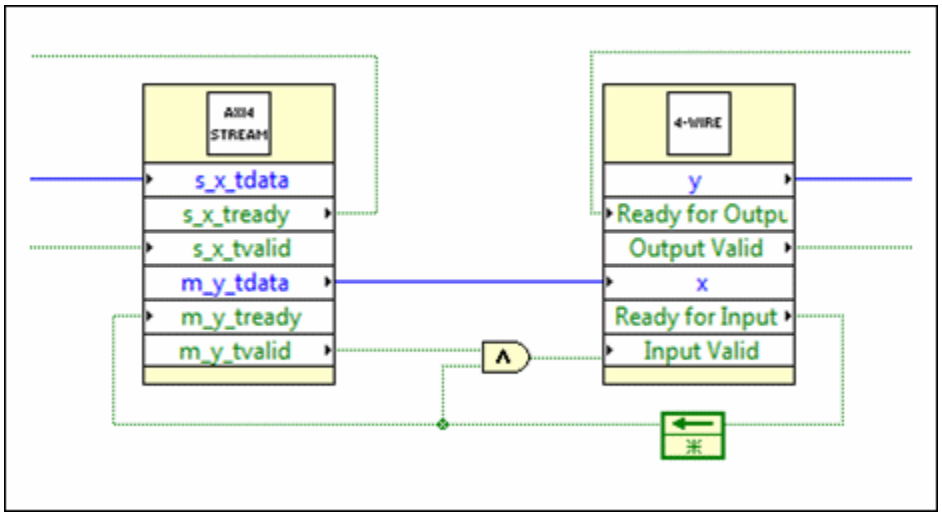
4线至AXI互连与AXI至AXI互连在寄存器的使用上类似。下列程序框图显示了连接4线IP块中的一条信号至AXI IP块的方法。



AXI至4线互连

AXI至4线互连与4线至4线互连在寄存器的使用上类似，但需要使用“与(AND)”门控以遵从4线协议要求，即仅当下方数据流块准备就绪输出时，可提供有效数据。如

下列程序框图所示。



相关概念：

- 集成第三方IP
- Xilinx IP列表
- 使用握手信号控制定时
- 在循环间存储数据

优化FPGA VI的执行速度和大小

如要优化FPGA VI的性能，可修改FPGA VI提高其速度，降低FPGA逻辑资源占用，等等。

下表包含FPGA VI的优化技巧。

注： 要理解表格中的各种技巧，您必须熟悉寄存器。

优化技巧	FPGA速度	FPGA大小
减少组合路径。	✓	
合适时使用流水线模式。	✓	
使用单周期定时循环。	✓	✓
使用并行操作。	✓	
选择 **从不仲裁**作为仲裁选项。	✓	✓
使用非重入子VI。		✓
使用重入子VI。	✓	

限制前面板对象的数量，例如，数组。		✓
使用最小可用的数据类型。	✓	✓
限制自定义数据类型的大小。	✓	✓
尽可能避免大型VI和函数。	✓	✓
使用握手信号控制定时。	✓	✓
访问DMA FIFO时使用外部数据值参考。	✓	✓
尽可能通过配置双端口读访问，减少块内存资源的使用。	✓	✓
除非需要使用其他类型存储器，否则请优先考虑使用块内存。存储器块不会消耗FPGA资源，且相对于其他类型的存储器倾向于使用高时钟速率执行编译。	✓	✓
从单周期定时循环内部移除隐式启用信号，该循环独立于程序框图上的其他节点运行。该策略主要用于大型设计。	✓	

相关概念：

- 管理共享的资源
- FPGA硬件概述
- 缩短FPGA VI的组合路径
- 使用流水线优化FPGA VI
- 使用单周期定时循环优化FPGA VI

- 使用并行操作
- 避免仲裁以优化FPGA VI
- 判定何时使用重入或非重入子VI
- 限制FPGA VI中顶层前面板对象的数量
- 使用最小的数据类型优化FPGA VI
- 使用带有寄存器项、存储器项、FIFO和握手项的自定义数据类型
- 在可能的情况下，在FPGA VI中尽量避免使用较大的VI和函数
- 使用握手信号控制定时
- 通过双端口读取降低存储器资源使用

缩短FPGA VI的组合路径

长的组合路径需要更久的执行时间，也限制了时钟域内的最大时钟速率。

由于输入寄存器和输出寄存器间的逻辑必须在指定的同一时钟周期内执行，在单周期定时循环中，长的组合路径通常存在问题。在单周期定时循环中，LabVIEW移除了组件内和组件间的寄存器，即增加了寄存器间的组合路径长度。如组合路径中的代码未在一个时钟周期内完成，LabVIEW将在编译状态窗口返回定时冲突。

注： 多层嵌套的条件结构也会导致LabVIEW在编译状态窗口返回定时冲突。

如要缩短组合路径的长度，首先应尽可能简化逻辑。逻辑简化为最简模式后，可通过划分逻辑为不同的级和流水线设计进一步缩短组合路径的长度。

注： 如在单周期定时循环内使用高吞吐率数学函数，用户可通过几种方法缩短函数间组合路径的长度。

相关概念：

- 在单周期定时循环中执行代码
- 实现多个时钟域

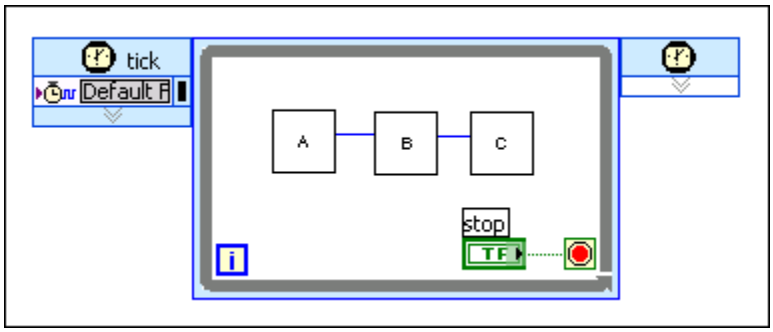
使用流水线优化FPGA VI

流水线是一种可用于增强FPGA VI时钟速率和吞吐量的技术。在流水线设计中，用户可利用FPGA的并行处理特性提高顺序代码的有效性。如要实现流水线，必须将代码拆分为不同的级并连线每级的输入和输出端至循环中的反馈节点或移位寄存器。

下文介绍了FPGA VI在单周期定时循环内的标准执行和流水线执行。

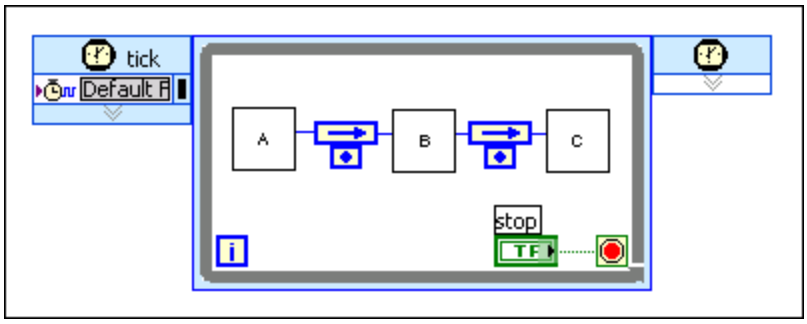
单周期定时循环中的标准执行

在下列程序框图中，子VI **A**、**B**和**C**在单周期定时循环内顺序执行。因此，单周期定时循环的时钟速率必须设置为满足上述三个个运行子VI的运行时间的和值。



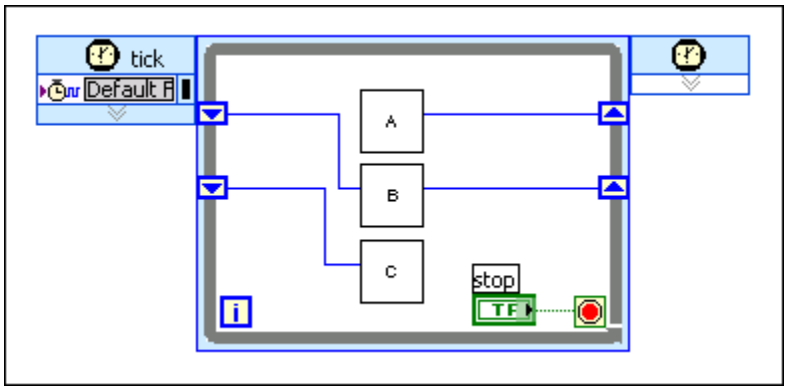
单周期定时循环中的流水线执行，使用反馈节点

在下列程序框图中，由于子VI的输入和输出连线至反馈节点，LabVIEW流水线处理子VI。在该FPGA VI中，子VI在单周期内并行执行，且最大时钟速率仅受具有最长组合路径的子VI的限制。



单周期定时循环中的流水线执行，使用移位寄存器

移位寄存器也可用于实现流水线代码，如下列程序框图所示。

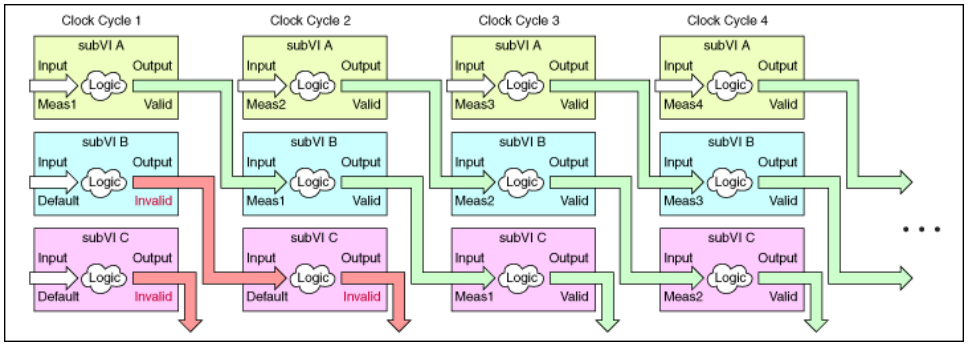


实现流水线代码

实现流水线代码时考虑下列操作：

- 最后一级的输出滞后输入的值等于流水线的级数。
- 流水线填满前，时钟周期的输出无效。
- 流水线的级数称为流水线深度。
- 流水线延迟（以时钟周期为单位）对应其深度。流水线深度为N时，第N个时钟周期前的输出无效，且每个有效时钟周期的输出比输入端延迟N-1个时钟周期。

请参考以下范例。



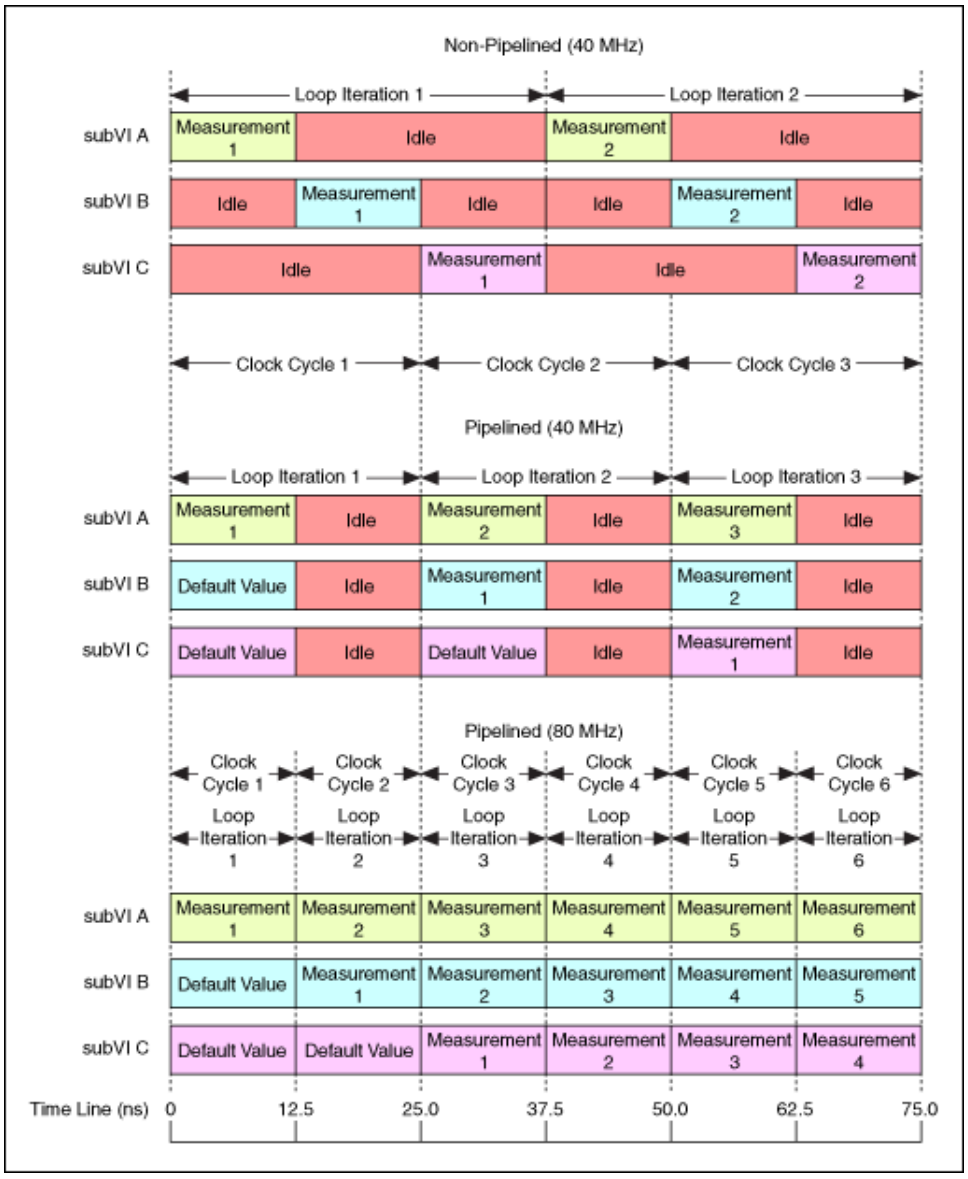
在该范例中，三个独立的执行步骤分别执行子VI **A**、**B**和**C**，即流水线深度为3。由于该代码需要三个执行步骤，输出要到**时钟周期3**才有效。每个有效时钟周期C的输出总是对应时钟周期C – (N – 1)的输入。

时钟周期	说明
时钟周期1	在**时钟周期1**中，**子VI A**处理第一个测量值(**Meas1**)，而**子VI B**和**子VI C**都处理移位寄存器的默认值(**Default**)，产生无效输出。
时钟周期2	在**时钟周期2**中，**子VI A**处理第二个测量值(**Meas2**)，**子VI B**处理**时钟周期1**中**子VI A**的输出，**子VI C**处理来自**子VI B**的无效输入，从而产生无效输出。
时钟周期3	在**时钟周期3**期间，由于所有输入都有效，并且**子VI C**的输出首次有效，流水线最终填满。**子VI A**处理第三次测量(**Meas3**)，**子VI B**处理**时钟周期2**中**子VI A**的输出，而**子VI C**处理**时钟周期2**中**子VI B**的输出，从而产生与第一次测量(**Meas1**)相对应的输出。流水线填满后，全部后续时钟周期均生成有效的输出，常量延迟为两个时钟周期。

提示 考虑使用条件结构避免无效输出导致的未预期的操作，并确保控制算法在N个时钟周期后启用执行器。

使用流水线增加吞吐量

使用流水线可增加吞吐量，因为流水线可在单周期定时循环内以更快的时钟域内运行。



非流水线 (40 MHz)

示意图顶部为非流水线循环的执行时间。该代码包含三个子VI，每个需要12.5 ns的传播延迟。**子VI A**至**子VI C**的全部延迟为37.5 ns，相对于40 MHz编译频率，延迟时间过长。

流水线 (40 MHz)

示意图的中部给出了流水线处理代码将传播延时减少至12.5 ns，从而循环可在40 MHz进行编译。

流水线 (80 MHz)

示意图底部为使用高达80 MHz时钟速率编译的循环，因为流水线循环的传播延迟仅为12.5 ns。

注： 流水线设计相对于非流水线设计，增加了时钟周期延迟。但由于流水线能够降低单个时钟周期的时间，因此总的时间延时不会发生显著变化。

相关概念：

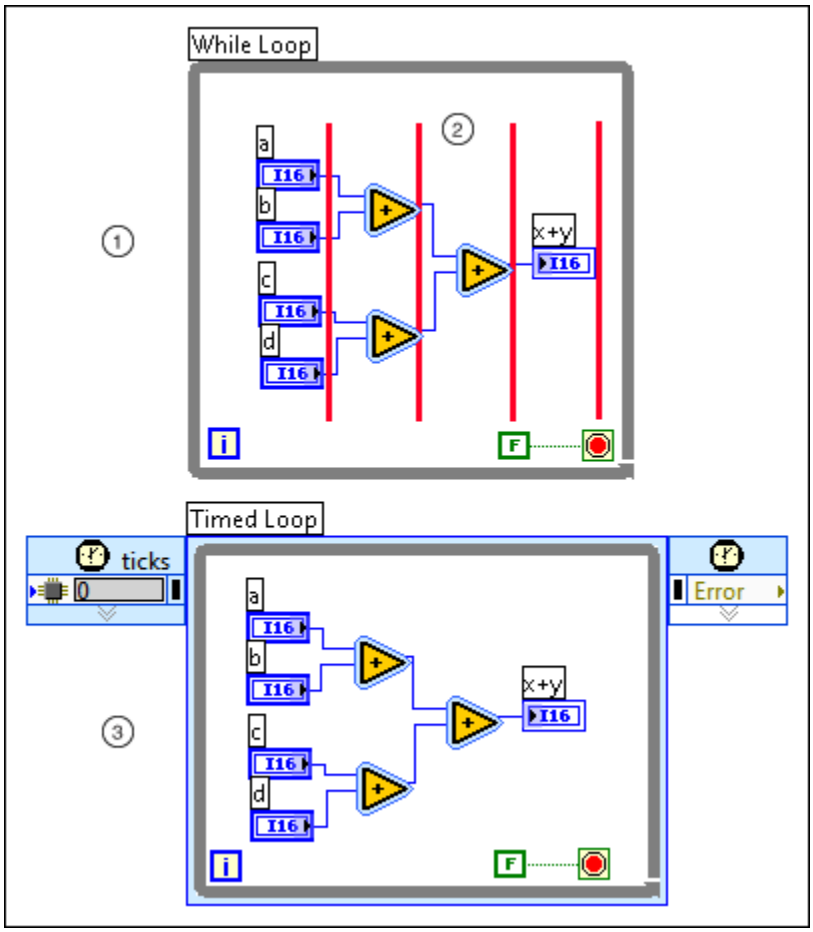
- 在单周期定时循环中执行代码
- 使用多个输入通道滤波FPGA I/O
- FPGA硬件概述
- 优化FPGA VI的执行速度和大小
- 实现多个时钟域
- FPGA VI的定时考虑因素
- 缩短FPGA VI的组合路径

使用单周期定时循环优化FPGA VI

LabVIEW自动优化单周期定时循环(SCTL)内的代码。与While循环内的相同代码相比，SCTL内的代码执行更快，占用FPGA终端资源更少。在FPGA终端上使用While循环时，While循环每执行一次需要占用多个时钟周期，While循环包含启用链寄存器。While循环每次执行需要占用的时钟周期的数量取决于循环内的代码。在FPGA终端上使用单周期定时循环时，单周期定时循环将在一个时钟周期内执行完循环内的所有代码。在FPGA目标上使用单周期定时循环减少了执行周期和资源占用，因为单周期定时循环不包括启用链寄存器。如果单周期定时循环包含已初始化的移位

寄存器，循环第一次执行前占用一个时钟周期，以初始化移位寄存器的值。单周期定时循环类似于HDL中的定时进程。

下列图示展示了执行相同代码的While循环和单周期定时循环之间的区别。

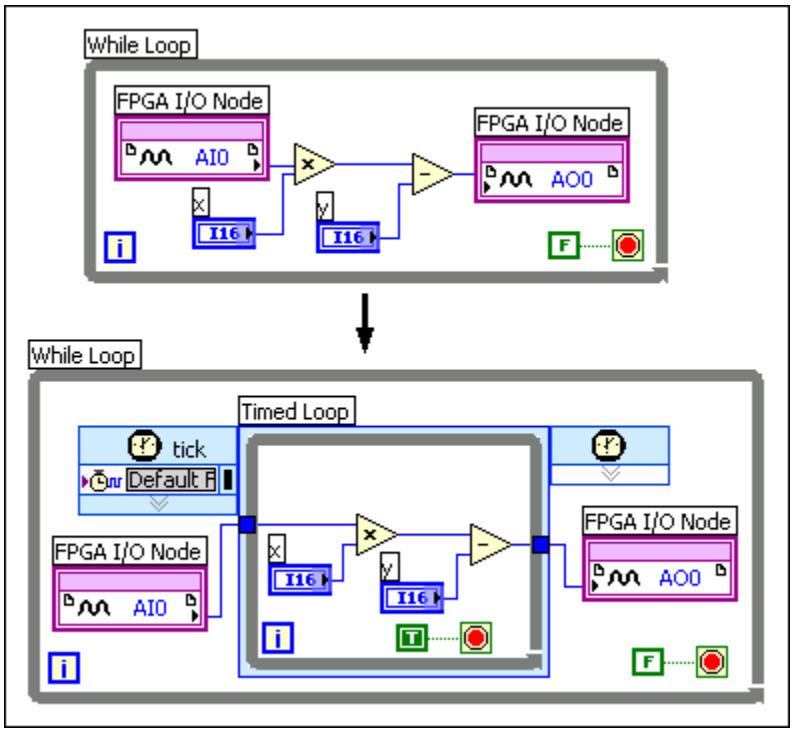


下面列出了上述程序框图的要点。

①	执行While循环内的代码需要四个时钟周期，不包括While循环占用的两个时间周期作为额外开销时间。
②	红色垂直线表示每个时钟周期内While循环执行的结束位置。
③	如时钟周期对于代码而言足够长，同样的代码放在单周期定时循环内，可以在一个时钟周期内执行完毕。

您也可使用单周期定时循环减少FPGA VI中的执行周期，以优化代码。如下图所

示。

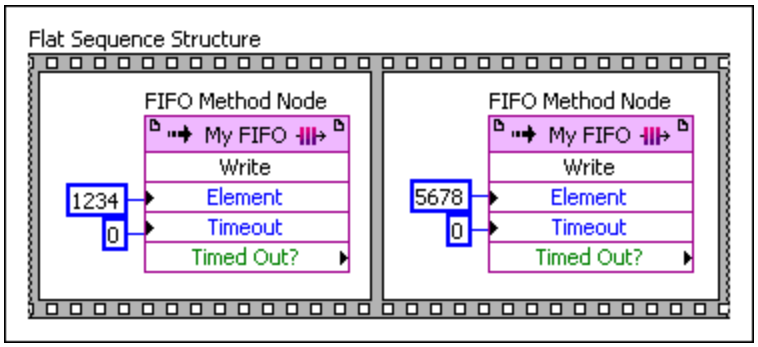


如上图所示，如果在While循环中使用单周期定时循环，则将TRUE常量连接到条件接线端，使得定时循环内的代码在While循环的每个周期执行一次。

避免仲裁以优化FPGA VI

LabVIEW使用仲裁机制来管理对共享资源的访问。选择仲裁选项时，请尽可能选择**从不仲裁**，节省FPGA的资源并提高速度。

如果您为资源接口选择**从不仲裁**选项，则LabVIEW不会添加仲裁组件，显著节约了FPGA资源。除了节省资源，**从不仲裁**选项还使一些FPGA I/O和FIFO函数可在单时钟周期内执行。要使用**从不仲裁**选项，您必须保证按顺序访问FPGA VI数据流中的资源接口，如下图所示。



在上图中，平铺式顺序结构确保两个FIFO方法节点同时执行，资源竞态不会发生。在这种情况下，**从不仲裁**是一个合适的选项。但是，如您选择**从不仲裁**选项并同时发出请求，FPGA VI将会损坏数据。

注： 为确保数据完整性，即使读取和写入不是同时发生的，也需避免多个对象同时发生FIFO读取或写入。

仿真包含多个处理器的存储器项的FPGA应用时，选择**从不仲裁**选项可能导致错误行为。例如，应用程序包含多个写入方，每个写入方可在仿真时更新指定的存储器地址。此外，如应用程序包含多个读取方，每个读取方可在仿真时读取指定的存储器地址。

相关概念：

- 理解仲裁选项
- 管理共享的资源
- 优化FPGA VI的执行速度和大小

限制FPGA VI中顶层前面板对象的数量

如要降低FPGA VI的空间使用量，可减少顶层FPGA VI的前面板输入控件和显示控件的数量。由于LabVIEW包含与顶层FPGA VI及主VI通信的其他VI，顶层FPGA VI中的每个前面板对象均占用大量的FPGA空间。存储前面板输入控件和显示控件数据的寄存器需要的触发器数量高于内部寄存器所需的触发器。

子VI中的前面板对象不直接与主VI通信，因此不会占用额外的FPGA空间。在一个调

用和另一个调用间必须保留状态信息的子VI输入控件及显示控件使用寄存器存储数据。仅传入和传出子VI数据的子VI控件不占用FPGA资源。

限制数组

由于数组的每一位均使用一个触发器，因此显示为顶层前面板对象的数组消耗大量的FPGA空间。考虑使用FIFO或存储器项替代数组传输数据。

连线数组以用作函数的输入时，FPGA编译器将创建同等的For循环，以顺序处理数组中的每个元素。连线簇以用作FPGA VI或函数的输入时，FPGA编译器将创建并行逻辑，以处理簇中的每个元素。数组和簇为递归关系，即如果连线数组组成的簇用作输入端，会并行处理该数组，且顺序处理数组的元素。

注： 在数组上执行运算可限制最大顶层或衍生时钟速率。如要最大化FPGA时钟速率，可通过处理单个数据点来替代数组。

使用全局变量

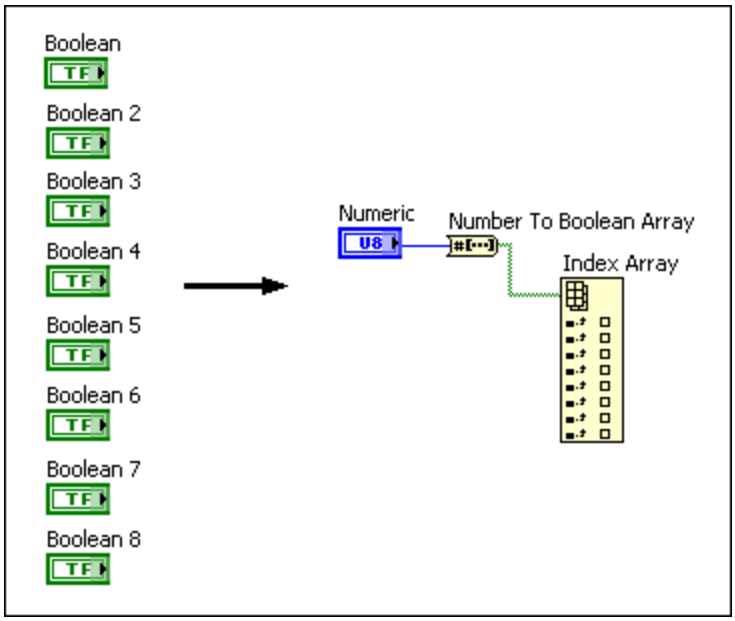
如不需要通过主VI访问前面板的输入控件和显示控件，可考虑使用全局变量替代输入控件和显示控件在FPGA VI间传输数据。如无需传输数据，可考虑使用常量替代控件，以降低FPGA VI空间的使用量。

使用反馈节点

如要降低资源使用量，可使用反馈节点替代控件在子VI中存储数据。

比特包数据类型

如要降低前面板对象的数量，可按照下列示意图组合对象。



在上面的示意图中，顶层FPGA VI的每个布尔控件均包含消耗额外FPGA资源的逻辑。但如使用单个8位数字控件表示数据，仅一个控件包含额外的逻辑资源。使用“数值至布尔数组转换”和“索引数组”函数获取数值控件的每个元素。

相关概念：

- FPGA硬件概述
- 优化FPGA VI的执行速度和大小
- 使用直接内存访问传输数据
- 使用前面板输入控件和显示控件传输数据

使用最小的数据类型优化FPGA VI

尽量为非常量值选择最小数据类型，以降低FPGA VI的大小并加快执行速度。创建FPGA VI时请考虑下列规范。

- 数值函数 – 数值函数强制转换所有输入数据为最大的数据类型。例如，如连线16位整数和32位整数至乘法函数的输入端，函数将强制转换16位整数为32位整数。如要避免强制转换为大的数据类型，可使用转换为定点函数转换整数输入为同等的定点数据类型，并为所需的定点表示法配置乘法函数输出端。
- 循环定时器、时间计数器和等待VI – 可以使用可用于FPGA VI的最小**内部计数器**

大小。

- While循环 – 可将32位计数接线端输出转换为最小的数据类型，且仍支持循环执行的最大循环次数。使用移位寄存器也可避免使用计数接线端。
- 索引数组函数 – 索引数组函数使用32位整数值作为**索引**输入端的默认值。如连线至**数组**输入端的数组包含小于256个元素，可将输入端的表示法由32位整数更改为8位整数。

为下列程序框图对象选择最小的可用数据类型，能够节省FPGA资源。

- 比较函数
- 连线至条件结构的条件选择器
- 输入控件和显示控件

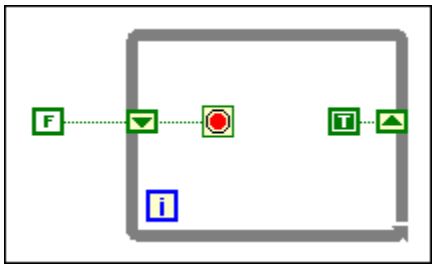
相关概念：

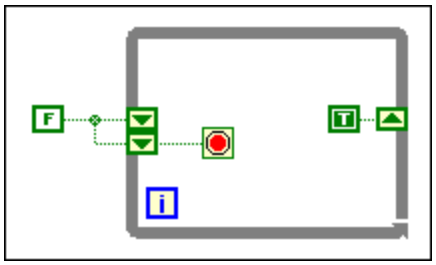
- 避免计数接线端数据类型
- 优化FPGA VI的执行速度和大小

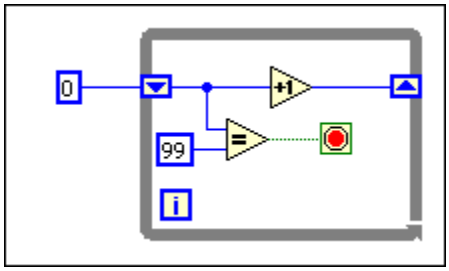
避免计数接线端数据类型

尽量为非常量值选择最小数据类型，以降低FPGA VI的大小并加快执行速度。使用While循环时，可使用移位寄存器以避免使用计数接线端的32位整数。

下列示意图显示了不使用计数接线端创建运行2、3和100次循环的While循环。







在最后一个示意图中用于初始化寄存器的常量为无符号8位整数数据类型。

相关概念：

- 使用最小的数据类型优化FPGA VI

在可能的情况下，在FPGA VI中尽量避免使用较大的VI和函数

如果要优化FPGA VI，请谨慎使用以下VI和函数：

- 商与余数－此函数占用大量的FPGA资源。如果除数是2的幂次方，可使用按2的幂缩放函数，且**n**输入端连接负常量。
- 除法和倒数－这些函数会消耗FPGA上的大量空间。但是，如果将舍入模式设置为**截断**或**半值向上（不对称）**，以节省FPGA空间。
- 平方根－此函数提供了一个默认的**sqrt(x)**输出，具有很高的精度。如果**sqrt(x)**的小数字长大于**x**的一半，则可以减少**sqrt(x)**的小数字长为较低的精度，以节省FPGA时钟周期和资源。
- 一维数组循环移位—如将控件连接至输入端，该函数所需的时间与移位的数量成正比，外加进入和离开函数的两个时钟周期。如连接常量至输入端，函数执行和时间和占用的FPGA资源可以忽略不计。

- 按2的幂缩放—如连接输入控件至输入端，该函数将占用FPGA上的大量空间。如连线常量至输入端，函数将不占用FPGA资源。

> **注：** 在可能的情况下，使用常量替代输入控件能够优化FPGA VI。但中间结构（例如，连线板、循环隧道和移位寄存器）会妨碍LabVIEW识别常量输入端。

相关概念：

- 优化FPGA VI的执行速度和大小

通过双端口读取降低存储器资源使用

对于某些应用，可通过在“存储器属性”对话框的“接口”页面配置用于双端口读取访问的存储器，减少块存储器资源的使用量和/或缩短执行时间。

双端口读取存储器的主要优点在于通过两个读取端口，用户可同时访问两个不同地址的数据。例如，同一正弦或余弦函数的系数。用户可将该函数在存储器块中保存一次，然后通过两个地址读取不同的相位值。两个读取该存储器的存储器方法节点可位于同一结构中（例如，单周期定时循环内）或FPGA VI的不同位置。

优化数组常量的内存使用量

通过指定LabVIEW在存储器中实现数组常量的方法，可优化FPGA应用。新的数组常量的默认执行为**自动**，即编译器根据具体的编程模式判定在存储器块、查找表或触发器实现数组常量。

如编译使用**自动**设置的FPGA VI时遇到困难，可选择**存储器块**或**查找表**实现数组常量。除非需要利用不同存储器类型的优势，否则考虑选择存储器块实现数组常量。存储器块不会消耗FPGA资源，且相对于其他类型的存储器倾向于使用高时钟速率执行编译。

在其他环境下创建的数组常量（例如，在开发计算机）在FPGA VI中打开时被自动

设置为**自动**。在FPGA VI中创建的数组常量被移至其他环境时，仍保留选中的存储器实现。

在存储器块或查找表实现数组常量的限制条件

为确保可在存储器块或查找表之一实现数组常量，必须满足下列条件：

- 不能连线数组常量至顶层显示控件。但可从不同子VI的“索引数组”函数分离出“替换数组子集”函数。
- 如在单周期定时循环内放置数组常量，该循环仅可包含一个“替换数组子集”函数和一个“索引数组”函数。对于每个上述函数，在每个循环计数仅可访问单个索引地址，且用户不能调整函数大小或包含多个值的读取/写入。
- 对于单周期定时循环内的存储器块实现，必须满足下列额外条件：
  - 必须经由“反馈节点”传输数组常量，该节点被配置为VI编译或载入时是全局初始化，FPGA VI重置时忽略初始化。如要全局初始化“反馈节点”，右键单击初始化接线端，选择**全局初始化»编译或加载时初始化**。如要“反馈节点”忽略初始化，在其属性对话框的**FPGA实现**页面勾选**忽略FPGA重置方法**。
  - 在读取和写入运算后必须放置“反馈节点”。
- 对于单周期定时循环内的查找表实现，必须连线数组常量至移位寄存器或反馈节点。

注： 如数组常量不满足上述要求，LabVIEW切换回触发器模式或返回错误。

用于数组常量存储器实现的编程模式

通过下列FPGA VI模式帮助用户确定数组常量使用选中的存储器类型。反之，如FPGA VI不能编译，使用下列模式帮助用户判定故障的原因。

注： 下列FPGA VI中的反馈节点包含一个黄色的感叹号，表明它们不能被异步重置、重置后可能包含错误数据及不能在顶层VI执行间重置。存储器块和查找表不支持“反馈节点”内部的异步重置状态。

单周期定时循环内的数组常量编程范例

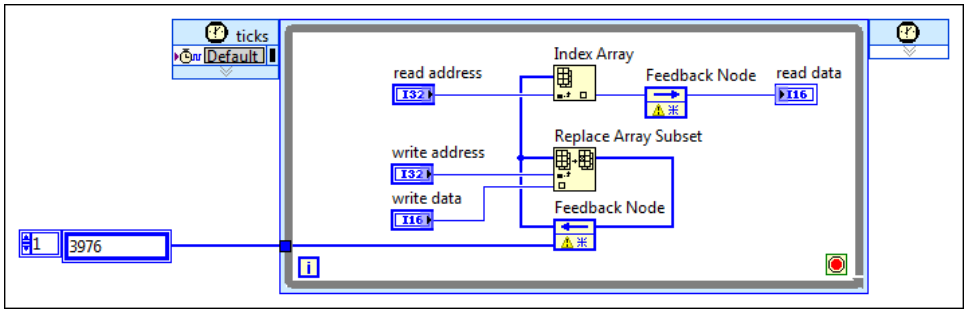
无

存储器块实现

如用户编程模式与下列范例类似，用户可在存储器块实现数组常量。

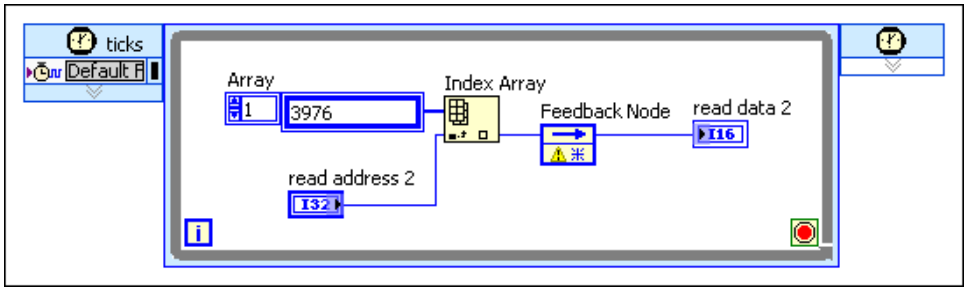
RAM模式

在下列编程模式中，注意“反馈节点”出现在“索引数组”执行的读取操作后，及“替换数组子集”执行的写入操作后。该模式可编译，因为其满足反馈节点仅有一个读取和写入操作的要求。



ROM模式

在下列编码模式中，注意“反馈节点”需放置在“索引数组”执行的读取操作后。

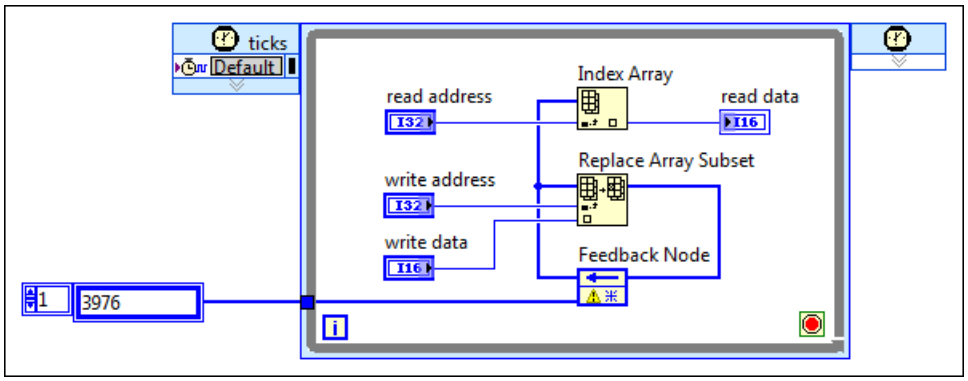


查找表实现

如用户编程模式与下列范例类似，用户可在查找表中实现数组常量。

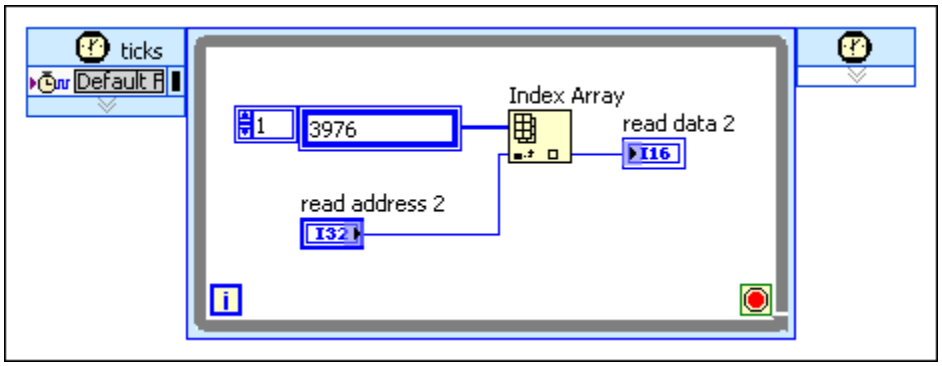
RAM模式

在下列编码模式中，注意在单周期定时循环内，无需在“索引数组”执行的读取操作后放置“反馈节点”。并且所需的“反馈节点”连线至数组常量。



ROM模式

在下列编码模式中，注意“反馈节点”无需放置在“索引数组”执行的读取操作后。

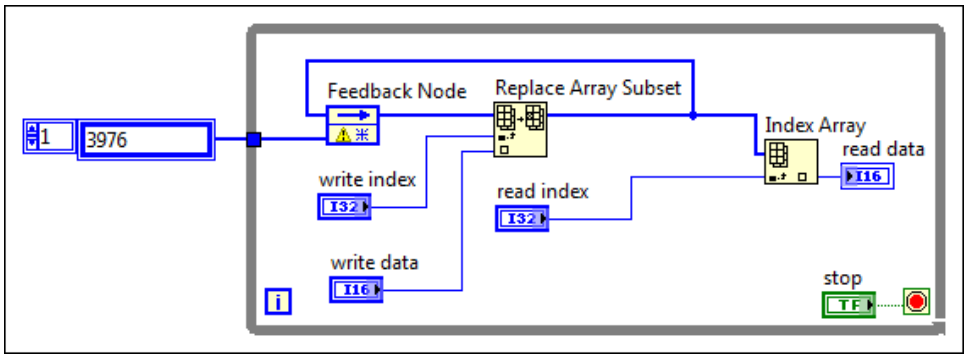


单周期定时循环外的数组常量编程范例

如用户程序代码与下列模式类似，可在存储器块或查找表中实现数组常量。

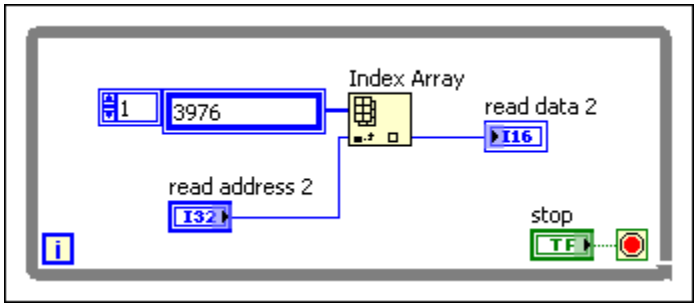
RAM模式

下列编程模式仅允许“替换数组子集”执行的一个写入操作，及其后续由“索引数组”执行的一个读取操作。



ROM模式

下列编程模式允许由“索引数组”执行的无限个读取操作，且无需使用反馈节点。



编译、下载和运行FPGA VI

下载和在FPGA终端上运行FPGA VI之前，必须先编译FPGA VI至一个位文件。下列章节列出了整个构建过程。

创建FPGA VI的程序生成规范

必须创建一个程序生成规范以编译FPGA VI至FPGA的位文件。程序生成规范规定了编译器创建位文件的方法。

按照下列步骤在**项目浏览器**窗口中创建程序生成规范。

1. 右键单击**项目浏览器**窗口中的**程序生成规范**，从快捷菜单中选择**新建»编译**，打开“编译属性”对话框。或者在**项目浏览器**窗口中右键单击现有的程序生成规范，从快捷菜单中选择**属性**显示此对话框。
2. 在对话框的信息页面指定程序生成规范的名称和其他描述性信息。
3. 打开源文件页指定顶层VI。FPGA VI仅可带有一个顶层VI。
4. 如FPGA终端支持，在Xilinx选项页面配置Xilinx生成选项。
5. 单击**确定**按钮关闭对话框或单击**生成**按钮开始编译FPGA VI。

编译FPGA VI

如要编译FPGA VI，必须配置FPGA终端的执行模式为在终端上执行FPGA VI。右键单击终端，从快捷菜单中选择**执行VI»FPGA终端**。

通过下列方式可编译FPGA VI：

- 单击**运行**按钮编译FPGA VI。如所用的FPGA终端支持交互式前面板通信，LabVIEW将自动在FPGA终端上运行FPGA VI。仅当VI或项目自最后一次编译VI后发生了变化，单击**运行**按钮编译VI。
- 右键单击**项目浏览器**窗口中的FPGA程序生成规范，从程序生成规范快捷菜单中选择**生成**或**重新生成**。

编译过程要经历几个阶段。编译FPGA VI可花费几分钟或几个小时。建议编译FPGA VI前测试和调试FPGA VI。

下载已编译的FPGA VI

编译FPGA VI后可在FPGA终端上下载和运行FPGA VI。

> **注：** 在一个FPGA终端上每次仅允许下载并运行一个FPGA VI。如在使用一个FPGA VI时尝试下载另一个VI至FPGA终端，LabVIEW将报错且下载失败。

通过下列方式可下载和编译VI。

- 右键单击**项目浏览器**窗口中的安装程序生成规范，从快捷菜单中选择**下载**以下载FPGA VI。
- 通过编程强制FPGA VI使用FPGA接口函数下载。
- 如终端支持交互式前面板通信，单击FPGA VI上的**运行**按钮。如FPGA VI为新建或发生了改动，编译结束后FPGA VI将自动编译和下载至FPGA终端。但如果VI已经位于FPGA终端，则LabVIEW不能下载FPGA VI。
- 如终端支持闪存，可将FPGA VI存储在闪存中。

如FPGA程序生成规范的编译属性对话框的信息页已勾选**加载至FPGA后运行**复选框，下载完成后，FPGA VI将在FPGA终端上自动运行。否则，下载FPGA VI后，必须手动在FPGA终端上运行FPGA VI。

运行已编译的FPGA VI

通过下列方式可运行VI。

- 使用FPGA接口运行FPGA VI。使用可编程式FPGA接口通信能够创建编程读取和写入FPGA VI前面板窗口的主控VI。
- 如终端支持交互式前面板通信，单击**运行**按钮运行VI。如使用交互式前面板通信运行FPGA VI，停止运行在FPGA终端上的VI前不能关闭FPGA VI。
- 如终端带有闪存，可通过闪存自动运行FPGA VI。

相关概念：

- 添加监控FPGA VI的显示控件
- 添加I/O至监控FPGA VI
- 使用FPGA程序生成规范
- 交互式前面板通信
- LabVIEW FPGA编译系统
- 下载FPGA VI至FPGA终端
- 下载FPGA VI至FPGA终端的闪存
- 使用主控VI与FPGA终端通信
- 通过闪存自动运行FPGA VI
- 远程编译FPGA VI
- 在未编译的情况下预估FPGA资源的使用情况
- 使用CLIP时钟
- 查看比特文件路径
- FPGA应用和项目简介

使用FPGA程序生成规范

编译FPGA VI至FPGA应用前必须创建一个程序生成规范。LabVIEW使用设置的程序生成规范选项从程序框图生成HDL和比特文件。

右键单击**项目浏览器**窗口中FPGA终端下的程序生成规范并选择下列选项。程序生成规范的可用选项随指定FPGA终端变化。

- **生成**－仅当FPGA VI或程序生成规范据上一次编译VI发生改动时生成当前文件。然后此命令将编译VI。
- **重新生成**－无论FPGA VI或程序生成规范是否发生改动均生成当前文件，然后编译VI。
- **预估资源的使用情况**－通过Xilinx工具在不编译的情况下预估FPGA资源的使用情况。如生成的代码签名非当前签名，此命令首先生成当前文件，然后预估资源的使用情况。可用选项随指定FPGA终端变化。
- **检查签名**－判定比特文件是否为当前文件。如比特文件不存在，则命令检查生

成的代码签名。
- **生成当前文件**－在无需编译VI的情况下生成当前文件。生成当前文件能够获取特定的代码生成错误。
- **显示编译结果**－显示编译状态窗口。必须创建程序生成规范以显示编译状态窗口。

注： 在**项目浏览器**窗口中单击每个程序生成规范，选择**生成**可同步创建多个程序生成规范。

指定默认的程序生成规范

默认的程序生成规范为**运行**按钮使用的编译及运行FPGA应用的程序生成规范。如单击**运行**按钮前未指定程序生成规范，LabVIEW将自动创建和指定相应VI的默认程序生成规范。

如要指定默认的程序生成规范，在**编译属性**对话框的源文件页面上勾选**设置为默认程序生成规范**。

相关概念：

- 编译、下载和运行FPGA VI
- 在未编译的情况下预估FPGA资源的使用情况

在未编译的情况下预估FPGA资源的使用情况

某些FPGA终端系列（例如，Virtex-5）可在未编译FPGA VI前估计FPGA VI的资源使用情况。LabVIEW使用Xilinx工具估计FPGA VI将要使用的逻辑片、LUT和触发器的数量。编译前估计资源的使用，帮助用户判定FPGA设计是否满足FPGA需求，而无需花费大量的时间等待编译完成。

注： 此资源估计未将用户在“Xilinx选项”页面中指定的选项考虑在内。

按照下列步骤估计FPGA应用的FPGA资源使用情况。

1. 右键单击FPGA的程序生成规范，从快捷菜单中选择**估计资源使用**，开始资源估计流程。如未显示**估计资源使用**菜单项，即终端不支持该功能。
2. 在“编译状态”窗口中跟进进程的状态。
3. 报表准备就绪后，从**报表**下拉菜单中选择**估计设备使用（预综合）**，显示该报表。

相关概念：

- 编译、下载和运行FPGA VI
- 编译状态窗口的可用报告
- 使用FPGA程序生成规范

LabVIEW FPGA编译系统

LabVIEW FPGA编译系统主要包含三个组件：LabVIEW、编译服务器和编译工作站。编译FPGA VI为可下载至终端上的FPGA芯片的位文件时，上述三个组件分别实现下列功能：

- **LabVIEW**－向编译服务器发送编译请求。
- **编译服务器**－接收来自LabVIEW的请求，并发送编译任务至可用的编译工作站。
- **编译工作站**－接受来自编译服务器的任务请求并编译FPGA VI。

如在同一计算机上安装LabVIEW FPGA模块和Xilinx编译工具，则无需对上述3个编译组件进行额外的配置。默认情况下，LabVIEW和编译工作站使用安装在本地计算机上的编译服务器`localhost`。如在远程计算机上安装Xilinx编译工具，可使用一个或多个远程编译工作站远程编译FPGA VI。

编译过程详解

编译时间取决于VI的大小、处理器速度以及编译计算机的可用内存量。计算机内存不够的情况下，较小的程序框图可能会快速完成编译，而较大的程序框图需要使用

大量的虚拟内存，进而导致编译失败或完成时间大幅超出预计。

下列步骤列出了编译FPGA VI的过程。LabVIEW在**编译状态**窗口显示编译状态。如已连接编译服务器，可查看不同的编译报告。

1. **生成中间文件**—LabVIEW将FPGA VI转换为发送至编译服务器的中间文件（HDL代码）。
2. **排序**－编译服务器排序任务并发送中间文件至编译工作站，以用于编译。
3. **HDL编译、分析和综合**－编译工作站转换中间文件（HDL代码）为数字逻辑元素。
4. **映射**－编译工作站在FPGA的物理块间拆分应用逻辑。
5. **布局和布线**－编译工作站分配逻辑至FPGA的物理逻辑块并在逻辑块间布线，以满足编译的资源或定时限制。
6. **生成编程文件**－编译工作站创建二进制数据，LabVIEW将这些数据存储在比特文件内。
7. **创建比特文件**－LabVIEW在项目文件夹的子目录内保存该比特文件。可在FPGA VI中下载和/或运行该应用。

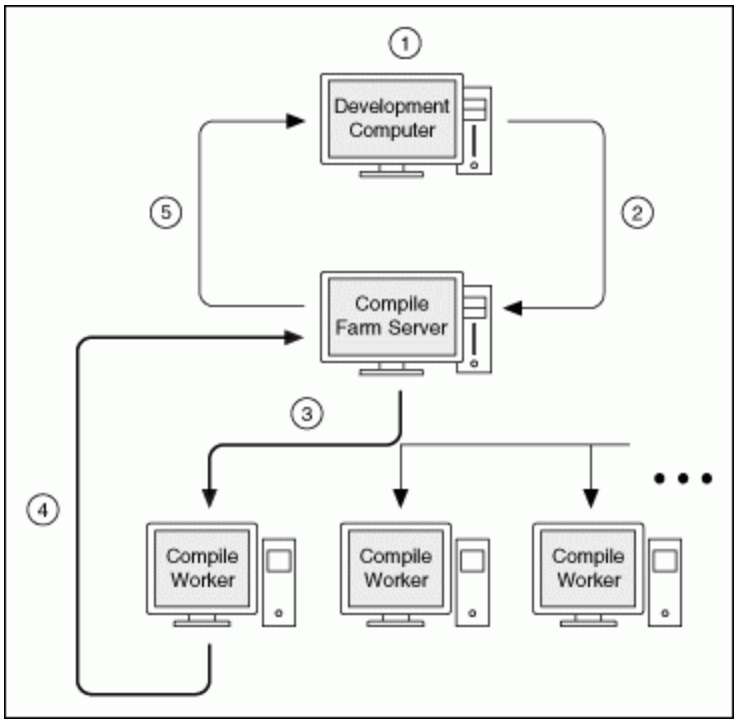
相关概念：

- 编译、下载和运行FPGA VI
- 远程编译FPGA VI
- 查看比特文件路径

Compile Farm入门指南

本主题介绍了创建和配置Compile Farm的方法。Compile Farm由一台现场编译服务器及一台或多台供多名开发人员使用的现场编译工作站组成。

下图及下文详细介绍了开发计算机、编译服务器及编译工作站将FPGA VI部署至FPGA终端的方法。



①	当前用户或其他开发人员在开发计算机上创建一个FPGA VI，并单击LabVIEW的**运行**按钮。
②	LabVIEW提交编译任务至编译服务器。
③	编译服务器发送编译任务至可用的编译工作站，由工作站编译FPGA VI。
④	编译工作站将已编译的VI返回编译服务器。
⑤	编译服务器将已编译的VI返回开发计算机。FPGA VI已准备就绪，可部署至FPGA终端。

软硬件要求和推荐

NI建议编译场的建立至少需要两台计算机：一台用作编译服务器，一台用作编译工作站。下表为上述计算机及开发计算机的说明。

注： 安装LabVIEW FPGA模块时，FPGA Compile Farm Server会自动安装到本地。如要使用远程编译服务器，必须在远程计算机上独立安装FPGA Compile Farm Server。关于安装FPGA Compile Farm Server的详细信息，见LabVIEW开发平台光盘上的FPGA模块安装选项。

创建Compile Farm

首先确定上表中提及的用作编译服务器、编译工作站及开发计算机的计算机，并确保全部计算机已连入同一网络。然后，按照下列步骤建立编译服务器、各编译工作站和开发计算。

建立编译服务器

按照下列步骤建立编译服务器。

1. 在要用作编译服务器的计算机上安装FPGA Compile Farm Server。记录计算机的IP地址或主机名称。
2. 登录编译服务器。
   a. 通过与编译服务器位于同一网络的另一台计算机，打开网络浏览器并导航至`http://<computername>:3580`，其中<computername>为之前记录的编译服务器的IP地址或主机名称。编译服务器显示NI基于Web的配置和监控的**系统配置**页面。通过该页面监控和管理队列中的编译任务。
   b. 单击**登录**。
   c. 输入NI基于Web的配置和监控的管理员权限的用户名和密码。默认情况下，用户名为`admin`，密码为空。

   > **注：** NI建议尽快为编译服务器设立安全机制。关于该任务的详细信息，在**安全配置**页面单击**帮助**，查看FPGA Compile Farm控制台帮助的保护终端章节。
   >
   > 通过编译服务器访问编译服务器时，无需登录。

3. 配置可使用Compile Farm编译FPGA VI的用户。
   a. 单击**安全配置**按钮。
   b. 单击**用户**选项卡底部的+按钮，添加用户名及分配密码。仅在该选项卡内列出的用户可使用Compile Farm。关于添加用户的详细信息，单击**帮助**查看控制台帮助的添加和删除用户章节。
   c. 分配一个或多个用户至**administrators**组。本组内的用户可使用Compile

Farm编译FPGA VI，还可以管理编译任务队列、取消编译任务和显示及隐藏编译工作站等。关于分配用户至"administrators"组的详细信息，单击**帮助**查看控制台帮助的为组指派用户，或从组中删除用户章节。

d. 完成添加用户和管理员后，单击**保存**关闭网络浏览器。

创建编译工作站

下一步为通知编译服务器每个可用的编译工作站。对每个编译工作站，执行下列操作：

1. 在要指定为编译工作站的计算机上安装Xilinx编译工具。
2. 配置编译工作站。
   a. 单击**开始»所有程序»National Instruments»FPGA»FPGA Compile Worker**打开编译工作站。Windows将在任务栏上显示**LabVIEW FPGA Compile Worker**图标  。
   b. 双击该图标，显示编译工作站的状态和性能。
   c. 单击**配置**配置编译工作站。
   d. 选择**连接至编译服务器**。
   e. 输入以下信息：
      - **主机名称**：`<computername>:3580`，其中<computername>为编译服务器的IP地址或主机名称。
      - **用户名**：`admin`
      - **密码**：与admin用户名关联的密码。默认状态下，密码为空。
   f. 输入**Number of simultaneous jobs**，即工作站可处理的同步编译任务的数量。通常，一个CPU内核可处理一个编译任务。但NI建议预留一个空闲内核以处理操作系统任务。例如，对于四核CPU，NI建议将**并发任务数量**设为3。
   g. 单击**确定**。
3. 验证是否每个编译工作站对于编译服务器均可见。
   a. 登录编译服务器。
   b. 单击**LabVIEW FPGA Compile Farm Console**按钮  。
   c. 验证添加的编译工作站是否出现在**Workers**列表中。可能需要修改“Filters”选项，以查看编译工作站。

设置开发计算机

配置开发计算机使用Compile Farm编译FPGA VI。对每台开发计算机，执行下列操作：

1. 安装LabVIEW、FPGA模块和NI-RIO驱动程序软件。
2. 在LabVIEW中，选择**工具»选项**，显示**选项**对话框。
3. 从**类别**列表中选择**FPGA模块**。
4. 定位至**编译服务器**，并选择**连接至网络编译服务器**。
5. 在弹出的对话框内单击+按钮添加服务器。
6. 在**主机名称**文本框中输入`<computername>:3580`。其中<computername>为编译服务器的IP地址或主机名称，然后单击**确定**。
7. 输入设置编译服务器时配置的**用户名**和**密码**。
8. 单击**测试连接**，测试开发计算机和编译服务器间的连接。如出现错误，修复错误。
9. 单击**确定**。

相关概念：

- 远程编译FPGA VI

断开FPGA编译服务器的连接

如要在VI编译期间关闭LabVIEW，可断开与编译服务器的连接。当FPGA VI编译或排队多个VI待编译时，无需断开LabVIEW与编译服务器的连接即可使用LabVIEW。

注： 请勿修改正在编译的FPGA VI或以下任意与FPGA VI关联的项。

- FPGA VI架构中的子VI
- **项目浏览器**窗口中FPGA终端下的任意项
- FPGA终端下的已初始化的组件级IP (CLIP)

按照下列步骤断开LabVIEW与编译服务器的连接。

1. 如未显示“编译状态”窗口，右键单击**项目浏览器**窗口中的程序生成规范，从快捷菜单中选择**显示编译结果**以显示该窗口。
2. 单击**编译状态**窗口中**关闭**按钮旁边的箭头，从下拉菜单中选择**全部断开连接**。**编译状态**窗口关闭。用户可选择继续运行LabVIEW或关闭LabVIEW。

通过闪存自动运行FPGA VI

有些FPGA终端带有闪存，可用来存储FPGA VI。关于闪存的详细信息见指定FPGA终端的硬件文档。

按照下列步骤自动运行FPGA VI，该VI在FPGA终端上电时由闪存载入。

1. 右键单击要在**项目浏览器**窗口中运行的FPGA程序生成规范，从快捷菜单中选择**属性**。打开编译属性对话框。
2. 在信息页面上勾选**载入FPGA时运行**复选框。
3. 单击**OK**按钮。
4. 配置FPGA终端在上电时自动载入VI。

> **注：** 如按上述步骤配置FPGA程序生成规范，配置FPGA VI的自动运行也会对打开FPGA VI引用函数的操作和下载方式产生影响。

相关概念：

- 编译、下载和运行FPGA VI
- 下载FPGA VI至FPGA终端的闪存

远程编译FPGA VI

可在非开发计算机上编译FPGA VI。如开发计算机运行缓慢或内存不足以编译FPGA终端的VI，用户可能想要在另一台计算机上执行编译。

默认情况下，LabVIEW使用安装在同一台本地计算机上的编译服务器和编译工作站。编译服务器随FPGA模块自动安装在本地计算机上。但必须在要用作编译服务器的远程计算机上安装FPGA Compile Farm Server，在用作编译工作站的计算机上安装Xilinx编译工具。

按照下列步骤配置LabVIEW，以在远程计算机上编译FPGA VI。

1. 在远程计算机上安装必需的Xilinx编译工具。
2. 浏览`FPGA\CompileWorker\`目录并运行`FPGACompileFarmConfiguration.exe`。
3. 勾选**Allow users to connect remotely to this compile server**复选框并单击确定按钮。
4. 浏览`FPGA\CompileWorker\`目录，打开编译工作站。

   > **注：** 必须在运行该工作站的计算机上打开编译工作站。不能远程打开编译工作站。

5. 使编译工作站保持在远程计算机上运行。
6. 在本地计算机上，打开包含待编译的FPGA VI的FPGA项目。
7. 选择**工具»选项**，可打开**选项**对话框。
8. 从**类别**列表中选择**FPGA模块**，显示**FPGA模块选项**对话框。
9. 在**编译服务器**部分选择**连接至网络编译服务器**。
10. 在**主机名称**文本框中键入运行编译服务器的远程计算机的名称或IP地址。
11. （可选）键入登录远程编译服务器的所需用户名和密码。默认情况下，**用户名称**为`admin`，**密码**文本框为空。
12. 单击**确定**。
13. 编译FPGA VI。如LabVIEW无法连接编译服务器，LabVIEW将在**编译状态**窗口显示错误信息。否则，每次编译FPGA VI时，LabVIEW在远程计算机上编译FPGA VI。

相关概念：

- 编译、下载和运行FPGA VI
- LabVIEW FPGA编译系统

- Compile Farm入门指南

相关信息：

- 在远程计算机上安装LabVIEW FPGA编译服务器
- NI LabVIEW FPGA编译选项

下载FPGA VI至FPGA终端的闪存

FPGA VI可被存储在指定FPGA终端的闪存上。关于FPGA终端闪存的详细信息见指定FPGA终端硬件文档。

按照下列步骤下载VI至FPGA终端的闪存

1. 新建项目或打开现有项目。
2. 添加FPGA终端至项目。
3. 在项目浏览器窗口，右键单击FPGA终端。从快捷菜单中选择**RIO设备设置**。此时将显示**RIO设备设置**对话框。

   > **注：** 单击**RIO设备设置**对话框中的**帮助**按钮可显示RIO Device Setup Help。

4. 在**待下载的位文件**路径控件中浏览或输入待下载至闪存的位文件的路径。
5. 单击**下载位文件**按钮。
6. 在**设备设置**选项卡中选择**设备上电时自动加载VI**或**设备重启时自动加载VI**。
7. 单击**应用设置**按钮。
8. 单击**退出**按钮。

> **注：** 即使未使用**加载至FPGA后运行**选项编译，下载至闪存的FPGA VI在FPGA终端上将自动载入FPGA。但VI不会自动运行。如要VI加载至FPGA后自动运行，请验证FPGA VI是否选择了**加载至FPGA后运行**编译选项。

下载VI至CompactRIO终端闪存的替代方式

用户可通过下列方式下载FPGA VI至CompactRIO终端的闪存。

- 按照下列步骤在不使用**RIO设备建立**对话框的情况下，下载VI至闪存。
  1. 右键单击要下载至闪存的VI，从快捷菜单中选择**下载VI至闪存**。此时将出现**确认将VI下载至闪存**对话框。
  2. 单击**是**按钮下载VI至闪存。
- 按照下列步骤，通过**开始**菜单下载VI至闪存。
  1. 浏览`NI-RIO\RIO Device Setup`目录，显示**RIO设备设置**对话框。
  2. 从**资源**下拉菜单中选中要下载位文件的FPGA终端。
  3. 在**待下载的位文件**路径输入控件中浏览或输入待下载至闪存的位文件的路径。
  4. 单击**下载位文件**按钮。
  5. 在**设备设置**选项卡中选择**设备上电时自动加载VI**或**设备重启时自动加载VI**。
  6. 单击**应用设置**按钮。
  7. 单击**退出**按钮。

相关概念：

- 编译、下载和运行FPGA VI
- 添加FPGA终端至LabVIEW项目
- 通过闪存自动运行FPGA VI

编译状态窗口的可用报告

FPGA VI编译期间，用户可在编译状态窗口中查看可用报表。通过报表信息判定FPGA VI是否满足FPGA要求和定时限制。**编译状态**窗口中的可用报表因具体的FPGA终端而异。

摘要报表详细信息

该报表包含生成的比特文件的摘要，且仅当编译结束后可用。如报表中包含定时错

误，可单击“定时冲突分析”按钮查看错误。

配置报表详细信息

该报表显示项目信息和用户在程序生成规范的“Xilinx选项”页面指定的Xilinx编译器配置。

估计设备使用（预综合）报表详细信息

LabVIEW使用Xilinx工具估计FPGA资源使用情况后，该报表可用。该报表是否可用随FPGA终端变化。报表包含下列信息：

- **设备使用** – 指示FPGA元素，如逻辑片、触发器、LUT和RAM块。
- **已用** – 指示编译FPGA VI使用的FPGA元素数。
- **总计** – 指示FPGA中的总FPGA元素数量。
- **百分比** – 指示FPGA应用程序使用的FPGA元素的百分比。如**百分比**大于100，将显示警告消息，提示用户估计设备使用已超出100百分比。根据所用的FPGA VI和硬件，Xilinx仍可能满足FPGA的需求。但用户可能需要优化FPGA VI。

估计设备使用报表详细信息

编译服务器完成编译进程的综合步骤后，该报表可用。报告包含FPGA VI综合期间对FPGA使用量的预估。报表包含下列信息：

- **设备使用** – 指示FPGA元素，如逻辑片、触发器、LUT和RAM块。
- **已用** – 指示编译FPGA VI使用的FPGA元素数。
- **总计** – 指示FPGA中的总FPGA元素数量。
- **百分比** – 指示FPGA应用程序使用的FPGA元素的百分比。如**百分比**大于100，将显示警告消息，提示用户估计设备使用已超出100百分比。根据所用的FPGA VI和硬件，Xilinx仍可能满足FPGA的需求。但用户可能需要停止编译并优化FPGA VI。

最终设备使用报表详细信息

编译服务器完成编译进程的映射或布局后，该报表可用。此报告为FPGA使用情况的总结，包含下列信息：

- **设备使用** – 指示FPGA元素，如逻辑片、触发器、LUT和RAM块。
- **已用** – 指示编译FPGA VI使用的FPGA元素数。
- **总计** – 指示FPGA中的总FPGA元素数量。
- **百分比** – 指示FPGA应用程序使用的FPGA元素的百分比。如**百分比**大于100，则编译失败。此时必须优化FPGA VI的大小。

估计定时报表详细信息

编译服务器完成编译进程的映射或布局后，该报表可用。报表包含FPGA VI映射或布局期间对FPGA时钟估计的总结。报表包含下列信息：

- **时钟** – 指示FPGA时钟。
- **所需值(MHz)** – 指示FPGA VI或FPGA VI组件能够运行的时钟速率（以MHz为单位）。某些FPGA VI组件（例如，定时循环）在程序框图上可见。某些组件（例如，CLIP）不可见。如**所需值 (MHz)**大于**最大值 (MHz)**，将显示警告消息，提示用户VI不符合定时限制。根据所用的FPGA VI和硬件，Xilinx仍可能编译FPGA VI，使其满足定时要求。但用户可能需要停止编译并优化FPGA VI。
- **最大值 (MHz)** – 指示FPGA VI或FPGA VI组件的理论最大编译速率（以MHz为单位）。

最终定时报表详细信息

编译服务器完成编译进程的布线步骤后，该报表可用。此报告为FPGA时钟的总结，包含下列信息：

- **时钟** – 指示FPGA时钟。
- **所需值(MHz)** – 指示FPGA VI或FPGA VI组件能够运行的时钟速率（以MHz为单位）。某些FPGA VI组件（例如，定时循环）在程序框图上可见。某些组件（例如，CLIP）不可见。如**所需值 (MHz)**大于**最大值 (MHz)**，编译失败。单击**调查定**

时冲突按钮，分析定时冲突。

- **最大值 (MHz)** – 指示FPGA VI或FPGA VI组件的理论最大编译速率（以MHz为单位）。对于指定FPGA终端，如设计满足定时要求，Xilinx编译器不会尝试优化FPGA VI。

Xilinx记录报表详细信息

该报表仅在编译结束后可用。如熟悉Xilinx工具，用户可能需要使用此文件解决编译失败问题。单击**保存**按钮，保存此信息至文件。

相关概念：

- 在未编译的情况下预估FPGA资源的使用情况