

CompactRIO 开发者指南

目录

第一章

概述及背景.....	1
开发指南概述.....	1
常用术语.....	1
机器控制构架概述.....	2
控制系统的配置.....	2
控制系统构架的结构图.....	3
Compact RIO简介.....	4
实时控制器.....	5
可重构的FPGA机箱.....	5
工业级I/O模块.....	5
Compact RIO规格说明.....	6

第二章

控制的基本架构.....	7
基本控制器架构的背景.....	7
初始化规则.....	7
控制规则.....	7
关闭规则.....	8
初级控制器架构例程.....	9
基于状态的程序设计.....	14
状态机概述.....	14
使用状态机的例程.....	14
Lab VIEW中的状态机例程.....	14
状态图表介绍.....	19
状态图.....	21
第一步：设计调用 VI.....	26
第二步：定义输入、输出以及触发.....	27
第三步：创建一个状态图.....	28
第四步：将状态图放置到调用VI里.....	28
开始执行—修改例子.....	30
第一步：修改I/O库.....	30

第二步：修改关闭程序	31
第三步：修改任务1来影射I/O	31
第四步：修改或重新编写状态表	31

第三章

开发可扩展性系统的软件技巧	32
重用函数	32
创建Lab VIEW里的可重用代码	32
创建可重用代码例程	33
Lab VIEW里的其他可重用代码	35
IEC 61131 功能块	35
多任务（多循环）	36
设置任务优先权及同步任务	36
在任务之间传递数据	37
触发任务	38
多循环系统的基于命令架构	38
用于命令的共享变量	39
数值命令	40
使用共享变量触发一个并行循环的例子	40
向I/O 扫描里添加数据	42
增加一个定制I/O 扫描任务(驱动循环)	42
优先权和定时	43
为定制I/O向内存表里添加入口	43
增加定制I/O 扫描逻辑	44
程序式I/O访问	44
读写I/O	45
配置I/O	46
在已配置的系统中发现I/O	46
数据记录	47
板载实时内存数据记录到TDMS文件中	48
初始化文件	48
将数据写入到文件里	49
读取数值	49

动态创建新文件.....	50
板载实时内存数据记录到ASC II文件.....	50
初始化文件.....	51
将数据写入到文件里.....	51
读取数值.....	51
动态创建新文件.....	52
整合数据记录程序与控制结构.....	53
检索记录的数据.....	53
错误和故障.....	54
故障引擎.....	54
使用错误处理循环的代码例子.....	55
实时监视器.....	56
硬件监视计时器.....	57

第四章

Compact RI0系统通信.....	58
通信综述.....	58
命令或消息型通信.....	58
过程数据型通信.....	58
数据流/缓冲型通信.....	59
使用网络发布的共享变量进行通信.....	59
网络共享变量背景.....	59
网络变量端子.....	60
共享变量引擎.....	60
发布订阅协议.....	60
网络发布的共享变量特征.....	61
管理和监视网络发布的共享变量.....	63
使用网络发布的共享变量来共享进程数据.....	64
使用网络发布的共享变量来发布命令.....	66
使用网络发布的共享变量的基于命令的构架例子.....	70
基于命令的高级构架.....	72
为命令使用网络变量需要考虑的重要事项.....	77
原始以太网（TCP/UDP）.....	77
创建自定义通讯协议.....	77

自定义通讯协议的例子.....	78
简单的信息协议（STM）—Lab VIEW执行程序.....	78
Compact RIO的串行通讯.....	80
RS232技术介绍.....	81
Lab VIEW的串行通讯.....	82
仪器驱动网络.....	84
Lab VIEW中串行通讯的例子.....	84
与PLCs（可编程逻辑控制器）或其他工业网络设备的通讯.....	86
工业通讯协议.....	86
Mod bus 通讯.....	87
Mod bus范例.....	88
Ethernet/IP.....	89
OPC.....	90
通过OPC从Compact RIO上发布数据.....	90
 第5章	
添加I/O至Compact RIO系统.....	93
添加I/O至Compact RIO.....	93
以太网I/O.....	93
步骤1. 构件扩展系统.....	93
步骤2 添加I/O至主控制器的扫描.....	94
步骤3 拷贝数据至内存.....	95
确定性以太网I/O.....	97
NI 9144简介-确定性以太网机箱.....	97
步骤1. 安装确定性扩展机箱.....	97
步骤2 添加确定性I/O至IO扫描.....	99
步骤3. 添加FPGA Intelligence至确定性扩展机箱.....	100
机器视觉/检验.....	102
机器视觉系统架构.....	102
照明和光学.....	103
软件选项.....	104
机器视觉/控制系统界面.....	105
使用Lab VIEW Real-Time的机器视觉.....	105
步骤1. 添加一个NI Smart Camera至Lab VIEW Project.....	105
步骤2. 使用Lab VIEW对NI Smart Camera编程.....	106

步骤3 与Compact RIO系统通讯.....	107
使用Vision Builder AI的机器视觉.....	108
步骤1. 使用Vision Builder AI配置NI Smart Camera.....	108
步骤2 配置检测任务.....	110
步骤3 与CompactRIO系统通讯.....	111
运动控制.....	112
运动控制器.....	112
LabVIEW NI SoftMotion及NI 951x驱动接口模块.....	113
CompactRIO上的运动入门指南.....	114
确定系统要求并选择组件.....	114
连接硬件.....	116
由LabVIEW Project配置控制器.....	117
使用LabVIEW NI SoftMotion API开发自定义运动应用.....	119
LabVIEW示例代码.....	124
 第六章	
通过LabVIEW FPGA定制硬件.....	126
通过LabVIEW FPGA扩展CompactRIO.....	126
使用LabVIEW FPGA的应用场合.....	126
FPGA概述.....	127
FPGA的优点.....	128
用LabVIEW FPGA编程.....	129
CompactRIO上的混合模式.....	130
例子—使用主机接口进行单点 FPGA/Real-time通信.....	131
FPGA编程基础.....	131
在实时程序中的主机接口与LabVIEW FPGA通信.....	133
例子-同步单点FPGA实时通信.....	135
例子-进行简单的单点FPGA实时通信.....	135
使用自定义I/O变量.....	135
自定义I/O变量.....	135
创建自定义I/O变量.....	136
例子—利用自定义I/O变量实现同步FPGA/实时通信.....	137
使用自定义I/O变量.....	137
Scan Clock I/O 项目.....	137

例子—利用DMA FIFO采集波形.....	137
配置FPGA与实时硬件之间的通信.....	137
模块下溢与不同采样模式的支持.....	138
主机同步与自动重启.....	139
板载标定与通道总数确认.....	139
使用实时应用程序读取DMA FIFO.....	140
使用Delta-Sigma-Based C系列模块采集波形数据.....	141
不含扫描模式的C系列模块.....	142
LabVIEW FPGA开发的最佳技巧.....	142
技巧1. 使用单周期的定时循环（SCTL）.....	142
技巧2. 使用模块化、可重用的子VI来编写FPGA程序代码.....	146
实时更新查询表（LUT）.....	149
技巧3: 编译前使用仿真技术.....	151
技巧4 同步循环.....	155
同步触发和锁存.....	156
技巧5: 避免“Gate Hogs”.....	157

第7章

为CompactRIO创建网络用户界面.....	162
使用LabVIEW建立用户界面及HMI.....	162
基本HMI架构背景.....	162
初始化及停止.....	162
I/O扫描循环.....	162
导航循环.....	163
Windows XP, XP Embedded及CE Operating 系统的基本HMI架构.....	165
I/O表格.....	165
初始化任务.....	165
I/O扫描循环.....	166
导航循环.....	167
入门指南-修改实例.....	173
步骤1.修改记忆表.....	173
步骤2. 修改命令类型定义.....	174
步骤3. 编辑I/O扫描循环.....	174
步骤4.编辑导航循环.....	174

第八章

系统的配置与复制..... 175

系统的配置..... 175

将系统配置在CompactRIO上..... 175

 将LabVIEW程序配置在易失性存储器上..... 175

 将LabVIEW程序配置在非易失性存储器上..... 176

在触模板中配置应用程序..... 178

 配置到触模板的链接..... 178

 将LabVIEW VI配置在易失性或者非易失性储存器中..... 179

 向嵌入Windows CE 式或者 XP式的目标配置可执行触模板应用程序..... 182

使用网络发布的共享变量来配置应用程序..... 182

 网络共享变量背景..... 182

 向拥有变量的目标上配置共享变量库..... 183

 卸载网络共享数据库..... 184

 配置共享数据库客户的应用程序..... 184

推荐使用的CompactRIO软件包..... 185

系统复制..... 186

具有实时复制功能的VI..... 186

创建复制功能..... 187

 NI-RIO系统复制工具..... 188

知识产权保护..... 188

锁定算法或者程序防止被复制或修改..... 189

 保护已配置的程序..... 189

 保护特殊的VI..... 189

 将程序代码锁定在硬件上以防止知识产权被复制..... 190

 得到CompactRIO系统的MAC地址..... 191

 CompactRIO信息检索工具..... 192

连接其他平台..... 192

 LabVIEW程序代码的可移植性..... 193

NI Single-Board RIO(嵌入式单板平台)..... 193

 将CompactRIO应用程序连接到NI Single-Board RIO或者R系列设备..... 194

附录 A

CompactRIO 入门指南..... 198

CompactRIO 入门教程..... 198

CompactRIO 及其组件.....	198
硬件安装与配置.....	198
在MAX中配置你的CompactRIO系统.....	200
添加CompactRIO系统到LabVIEW项目中.....	202
修改一个既有的LabVIEW项目.....	203

附录B

LabVIEW调试工具.....	204
LabVIEW开发应用调试.....	204
调试已部署的应用程序.....	205
Lab VIEW 分析工具.....	208
分析VI和代码段.....	208
监测Real Time中的目标资源.....	209
NI Distributed System Manager and Real-Time System Manager.....	209
调试Lab VIEW FPGA.....	212
开发FPGA与开发Lab VIEW Real-Time有何不同.....	212
Lab VIEW FPGA “后编译” 调试技术.....	212
位于开发计算机上的FPGA性能模拟.....	213

附录C

大型程序开发.....	215
最佳实践.....	215
大型程序开发指南.....	215
应用结构化软件工程思想.....	215
获取需求和管理.....	215
架构规划与设计.....	216
开发.....	216
验证与审查代码.....	218
部署.....	219

第一章

概述及背景

开发指南概述

如果您的NI Compact RIO控制器使用NI Lab VIEW Real-Time Module(实时模块) 8.6或者更高的版本, 那么您就可参考该指南提供的构架或建议来编写控制应用程序。本指南包含了Compact RIO新特征的使用方法, 比如NI扫描引擎、故障处理机制以及分布式系统管理器。在LabVIEW8.6中对这些特征也进行了介绍。Compact RIO内置的组件能使使用者更容易地去设计控制应用程序。另外, 这些相同的基本构架也能应用于其他硬件平台, 比如Compact Field Point、PXI以及基于Windows的控制器。Lab VIEW Real-Time是一套完整的编程语言, 它为开发者提供了多种方法来创建制程序, 另外它还能帮助您创建非常灵活的高级系统。在现实世界中Lab VIEW Real-Time控制器被应用到了很多领域, 比如核电站燃料棒的控制、测试发动机电子控制单元使用的半实物仿真、石油钻井的自适应控制以及高速振动检测的预测性维护等。本指南为工程师设计工业控制应用程序提供了一个框架, 另外本指南也可作为标准Lab VIEW Real-Time培训的辅助指导。本指南将指导您学习怎么创建Lab VIEW Real-Time应用程序, 使其不仅包含 PLC (可编程逻辑控制器) 的一般特性, 也包含处理非传统应用程序的方法, 比如高速缓存I/O, 数据记录以及机器视觉技术。

常用术语

只要您理解了下文所描述的实时编程及控制程序的基本概念, 您就可以使用Lab VIEW Real-Time以多种方法来创建控制应用程序。

- **响应**— 一个控制应用程序需要对一个事件作出反应, 比如I/O变换、HMI (人机交互界面) 输入或内部状态改变。从事件发生到控制程序响应所需要的时间, 就称为响应。不同的控制应用程序具有不同的响应容差, 从几毫秒到几分钟不等。大多数的工业应用程序要求将响应控制在毫秒到秒的范围内。响应设计是控制应用程序的一个关键设计, 因为响应决定了控制循环的速率, 并且它还影响了I/O, 处理器以及软件的选择。
- **确定性及抖动**— 确定性是一个控制循环的定时可重复性。抖动是发生定时错误时, 衡量确定性的方法。比如, 设定一个循环, 使其每50ms运行一次, 同时更新其输出值, 但有时候程序会运行50.5ms, 那么这个抖动就是0.5ms。高确定性和高可靠性是实时控制系统的主要优势, 同时高确定性是控制应用程序稳定运行的关键。较低确定性会使模拟控制变得杂乱无章, 同时也使系统反应迟钝。
- **优先权**— 大多数的控制器使用一个单处理器来处理所有的控制、监视以及通讯任务。因为使用一个处理器来处理多个并行的命令, 所以必须找到一个方法使处理器首先处理最重要的命令。给关键控制循环设置一个高的优先权, 这样就能得到一个拥有所有功能的控制器。这个控制器不仅能满足上面的要求, 还能保持原来的高确定性和快速响应。比如, 在一个拥有温度控制循环和嵌入式记录功能的应用系统中, 将循环设置为最高权限来取得记录操作的优先权, 并提供确定性的温度控制。这样就能确保那些低优先权的任务, 比如记录、网络 服务器、HMI等, 不会影响模拟控制或者数字逻辑。

机器控制构架概述

机器控制系统一般包含HMI和实时控制系统。实时控制器提供可靠及可预测的机器行为，而HMI为操作员提供一个图像化的使用界面来监视机器的状态以及设置操作参数。在一般的机器控制系统中，使用PLC（可编程逻辑控制器）或者PAC（可编程自动控制器）来执行控制系统。基本控制器包含以下功能：

- 模拟和数字 I/O
- 共享I/O或者变量值的记忆表
- 定义机器行为的序列引擎

除了支持这些PLC类的功能外，NI公司的PAC还能支持更多的高级功能。这些功能如下：

- 高速数据采集及分析
- 运动控制
- 视觉/检验
- 自定义的基于硬件的信号分析
- 数据记录

可以在使用Windows的PC上或者使用嵌入式OS的触摸式计算机上来编写HMI。HMI一般包含一下特征：

- 触摸屏操作
- 具有导航控件的分页显示系统
- 数据输入工具（按钮、键区等）
- 警告/事件显示或者记录

控制系统的配置

最简单的机器控制系统由一个控制器组成（如图1.1），这个控制器没有配置显示界面。这个控制系统一般用于那些除了维修或检测外不需要HMI的应用系统中。



图1.1没有显示界面的控制器

较复杂一些控制系统中增加了一个HMI或者更多的控制器（如图1.2）。这个控制系统一般用于由操作人员控制的机器中。



图1.2局部机器控制系统

复杂的机器控制系统中可能包含许多控制器或HMI（图1.3）。这个系统中一般拥有一个高级的终端服务器作为数据记录及转发的引擎。可以在大型或复杂的机器中使用这个控制系统。通过这个系统，可以与不同位置的机器进行通讯或者为不同的操作员分配不同的监视及控制任务。



图1.3分布式机器控制系统

控制系统构架的结构图

一个拥有PAC及HMI的控制系统包含所有的软件单元。使用这些单元，可以创建大多数的机器控制应用程序。学习创建基础的PAC系统，然后利用HMI将其扩展到其他机器控制系统中。

1. 一个控制器必须包含能与外部设备进行通讯的接口，比如传感器、执行器、HMI以及网络设备
2. 将当前数据储存在记忆表中（有时候也叫标签引擎）
3. 运行逻辑来控制机器或执行处理任务
4. 执行内部任务，比如启动
5. 监视及报告系统错误

HMI具有相似的单元。它除了执行控制外，还提供一个用户界面。可以在控制器或者HMI上执行额外的任务，比如警告、时间监测及记录。



图1. 4局部机器控制结构的概览

通过分析控制器运行原理，可以将控制系统分为更小的单元，每个单元负责处理一个特定的任务。图1.5展示了控制器的构架及机器控制应用程序的独立单元。在这些单元中，有一些是已经存在的，可以被机器控制构架直接引用，但是还有一些作为特定机器控制应用程序的一部分需要被开发出来。

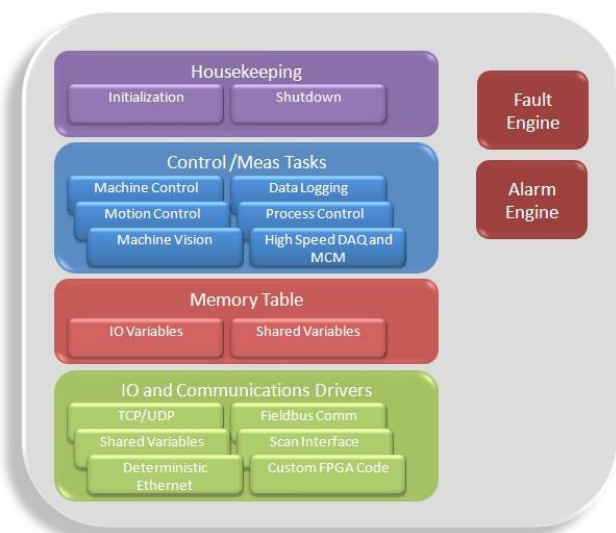


图1.5控制器构架及单元

本指南将逐一介绍各种控制器及HMI构架的推荐编程方式。同时还提供了一些实例的代码、不同的实现方法以及在这些实现方法之间的折衷考虑。

Compact RIO简介

Compact RIO是一款坚固耐用、可重配置的嵌入式系统，主要由三个部分组成——实时控制器，可重配置的FPGA（现场可编程门阵列）和工业级I/O模块。

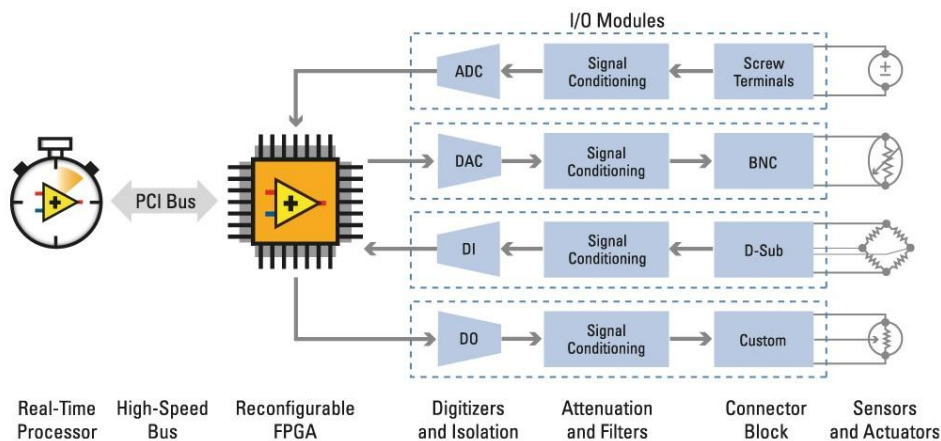


图1.6可重配置的嵌入式构架

实时控制器

实时控制器包含一个工业级处理器，能够可靠而准确地执行Lab VIEW实时应用程序，并可提供多速率控制、进程执行跟踪、板载数据存储以及与外部设备通讯等功能。另外，实施控制器还配备9~30 VDC的冗余电源输入、一个实时时钟、硬件监视定时器、双以太网端口、高达4GB的板载数据存储，以及内置的USB和RS232接口。



图1.7 NIcRIO-9014实时控制器



图1.8可重构的FPGA机箱

可重构的FPGA机箱

内嵌FPGA的可重配置机箱是嵌入式系统体系结构的核心。机箱中的FPGA直接和每个I/O模块相连，可高速访问I/O电路并灵活实现定时、触发和同步等功能。因为每个IO模块直连FPGA，而非通过总线，所以与其他工业控制器相比，Compact RIO几乎没有控制系统的响应延迟。默认情况下，该FPGA自动与I/O模块通信，并提供确定性I/O给实时处理器。您也可以直接对FPGA进行编辑来运行自定义的程序代码。由于FPGA的运算速度很快，内嵌FPGA的机箱经常用于构建具备高速缓冲的I/O、高速控制循环，或自定义信号滤波的控制器系统。例如，利用FPGA，单个机箱可以在100kHz的速率下同时执行超过20个的PID控制闭环。此外，因为FPGA代码最终映射为硬件逻辑，它的高可靠性和确定性非常适合实现硬件互锁，自定义硬件定时和触发这些功能，往往可以免去连接专用传感器所需的定制电路板制作。

工业级I/O模块

I/O模块包含隔离、转换电路，信号调理功能，并可直接与工业传感器或执行机构相连。通过提供多种连线选择并将连接器的接线盒








集成到模块上，Compact RIO系统大幅降低了对空间的需求和现场布线的成本。您可以从50多种NI C系列I/O模块中进行选择，它们使Compact RIO几乎能够连接所有的传感器或执行单元。模块的类型包括热电偶输入，±10 V同步采样的24位模拟I/O，24 V工业数字I/O（带有高达1 A的电流驱动），差分/ TTL数字输入，24位的IEPE加速度计输入，应变测量，RTD测量，模拟输出，功率测量，CAN连接和用以数据记录的SD卡等。此外，该平台是开放的，因此您可以自己开发应用模块或从其它厂商购买所需的模块。通过NI cRIO-9951 Compact RIO模块开发工具包，您可以开发自定义的模块以满足特定的应用需求。该工具包可使用户直接访问到Compact RIO嵌入式系统构架的底层资源以设计专用I/O、通信和控制等模块并包含Lab VIEW FPGA库以驱动自定义模块的接口电路。



图1.9可以选择50多种I/O模块来将Compact RIO连接到几乎所有的传感器和执行器上

Compact RIO规格说明

很多基于Compact RIO开发系统的客户将他们的系统销售并部署到世界各地。为了帮助用户简化部署地所需的系统设计过程，Compact RIO已通过了许多第三方机构的测试认证。



Description	Standard
Electromagnetic Compatibility (EMC)	89/336/EEC EN 55011 Class A at 10 m FCC Part 15A above 1 GHz Industrial levels per EN 61326-1:1997 + A2:2001, Table A.1 CE, C-Tick, and FCC Part 15 (Class A) Compliant
Product Safety	73/23/EEC EN 61010-1, IEC 61010-1 UL 61010-1 CAN/CSA C22.2 No. 61010-1
Hazardous Locations, Class I, Division 2	Class I, Division 2, Groups A, B, C, D, T4; Class I, Zone 2, AEx nC IIC T4, EEx nC IIC T4
Shock and Vibration	IEC 60068-2-64, IEC 60068-2-27, IEC 60068-2-6
Mean Time Before Failure (MTBF)	Bellcore Issue 6, Method 1, Case 3 MIL-HDBK-217F
Marine	Lloyds Register (LR Type Approval System Test Spec No. 1)
Quality/Environmental Management System (QMS/EMS)	ISO 9001/14001

Typical Certifications – Actual specifications vary from product to product. Visit ni.com/certification for details.

图1.10CompactRIO说明书

第二章

控制的基本架构

基本控制器架构的背景

建立复杂的系统需要一个可以实现代码重用、拥有可扩展性和运行管理的架构。以下两章将讲解如何建立一个用于控制的基本架构和如何利用该架构实现一个简单的PID循环。

一个基本的控制器架构包括三个主要状态：

1. 初始化(Housekeeping)
2. 控制(IO and Comm Drivers, Memory Table, Control and Meas Tasks)
3. 关闭(Housekeeping)

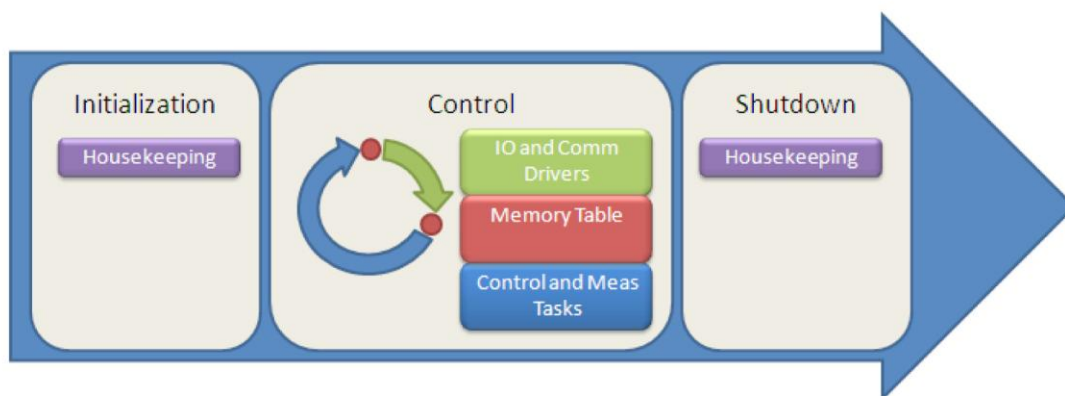


图2.1 基本控制器架构的三个主要状态

初始化规则  内部任务

在运行主控制循环之前，程序需要执行初始化程序。初始化程序使控制器处于运行的准备状态，但并非用于执行逻辑操作，如机器的启动或初始化。这些逻辑操作应在主控制循环中运行。初始化规则是：

- 1 设置所有内部变量为缺省状态。
- 2 创建必要的程序结构。包括队列、实时的FIFOs，VI refnums以及FPGA bit file下载。
- 3 执行额外的用户自定义逻辑使控制器处于运行的准备状态，如初始化日志等。

控制规则   

I/O、通信和内存表

很多程序员对直接I/O操作很熟悉，可以直接从硬件上发送及接受输入输出信号。这种方法对较小的单点应用下的波形采集及信号处理是理想的。但是，使用单点读写的控制程序在需要操作I/O的多重状态下会占用很大的资源。I/O操作增加了系统的总开销使系统变慢。而且，在一个程序中的各个层次管理多重I/O操作会使得程序难以改变I/O和执行模拟及强制动作。为避免这些问题，控制规则使用了Scanning I/O构架。在这种类型构架中，只能在每次循环间隔中进入物理硬件使用I/O及通信驱动(如图2.1所示)。输入输出值存储在内存表中，控制和测量任务在内存表中完成而不是直接进入硬件操作。这种架构有很多优点：

- I/O分离，程序员可以重复使用子VI和函数(无需硬件I/O代码)
- 较低的系统总开销
- 确定性运行
- 支持模拟
- 支持“强制”(forcing)
- 在逻辑执行中消除I/O改变带来的风险

控制和测量任务

控制和测量任务是定义控制应用的特定机器逻辑。可以是过程控制或更复杂的机器控制。在很多情况下，控制和测量任务基于状态机来处理多种状态的复杂逻辑。后面的章节将讲解如何使用状态机来设计逻辑。为在控制构架中执行，主控制任务必须：

- 比I/O 扫描速率运行更短的时间
- 通过I/O内存表存取I/O，而不直接读写I/O
- 除非在寄存器中保留状态信息，否则不使用“while循环”
- 除非在算法中，否则不使用“for 循环”
- 不使用“等待”而使用定时“timer”函数或“Tick Count”
- 不执行波形表(waveform)、记录(logging)或不确定性操作(使用并行的低优先权的循环来执行)

用户逻辑可以

- 包括单点运行如PID或逐点分析
- 使用状态机建立代码构架

用图解法表示，控制规则可以理解为是一个I/O读写和控制任务通过内存表通信运行的循环。当然，实际上，控制规则是多个同步循环和多个控制测量任务。

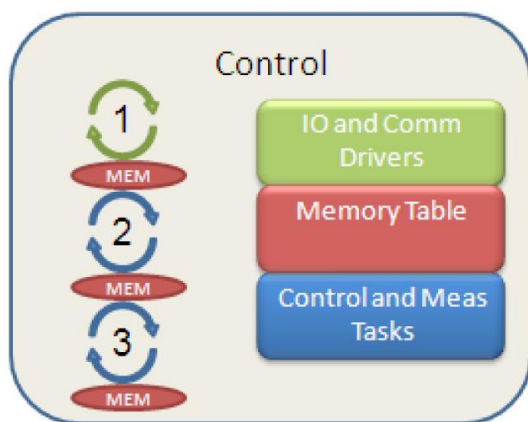


图2.2基础控制器的三大组成部分

关闭规则 **Housekeeping** 内部任务

由于收到命令或错误，控制器需要停止运行，此时停止主控制循环执行关闭规则。关闭规则使控制器停止运行并使其处于安全状态。关闭规则仅试用于控制器关闭而非用于机器关闭，后者应在控制循环中运行。关闭规则有

- 1 设置所有输出为安全状态
- 2 停止正在运行的所有并行循环
- 3 执行额外的逻辑动作如提示控制器操作人错误及日志状态信息。

初级控制器架构例程

在此，建立一个初级的PID控制例程。该例程控制一个温控室，维持350F。使用一个热电偶输出模拟信号，一个连接在加热器上的脉宽调制(PWM)数字输出，该例程使用PID控制算法。为了阐述整个控制家口，这个简单的例程没有增加复杂的控制结构。更详细的控制案例将在本书的后面章节一一介绍。

在Lab VIEW中建立这个案例，使用了5个控制器架构单元：

- 1 初始化规则
- 2 关闭规则
- 3 一个简单的过程控制任务
- 4 内存表中的 I/O变量
- 5 操作 I/O的RIO Scan界面

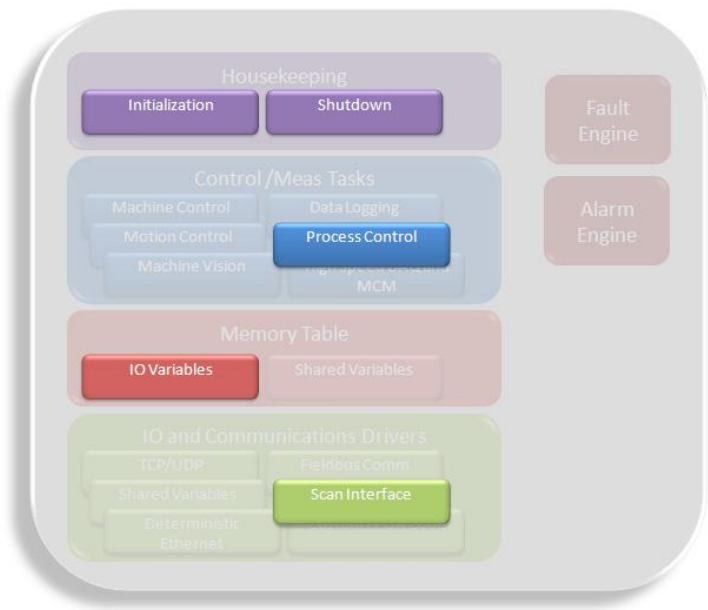


图2.3PID控制器构架

初始化和关闭规则

- 1 首先添加初始化和关闭规则。初始化规则需要配置控制器使其处于执行逻辑运行的准备状态，关闭规则需要执行基于关闭状态的动作。
- 2 为管理控制器运行顺序，建立一个有三个帧的顺序结构，分别是：初始化规则、控制和测量任务、关闭规则。

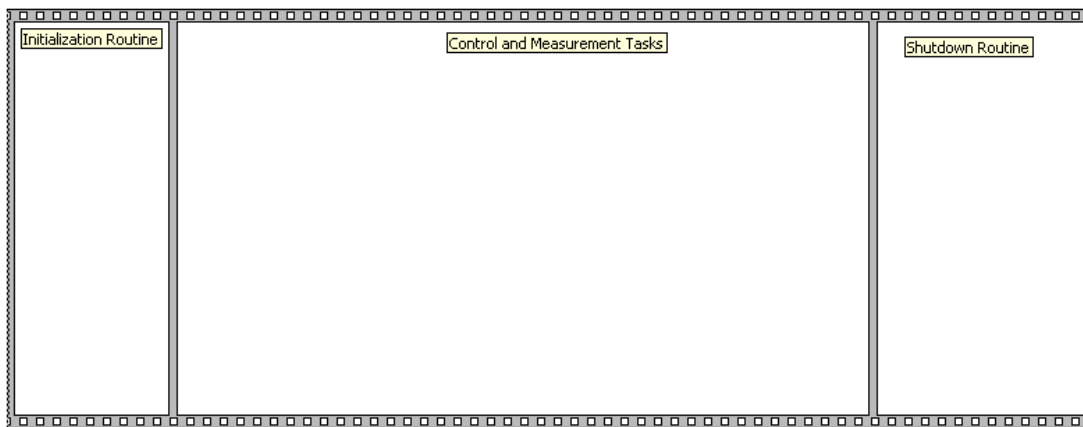


图2.4将控制结构分为三部分：初始化、控制和测量以及关闭

- 3 添加初始化和关闭逻辑操作。在此案例中，控制器无需初始化，控制器保留最后状态下的输出值。在此案例的关闭规则中，需要设置输出为关闭状态。我们可以增加其他逻辑操作如错误日志等。

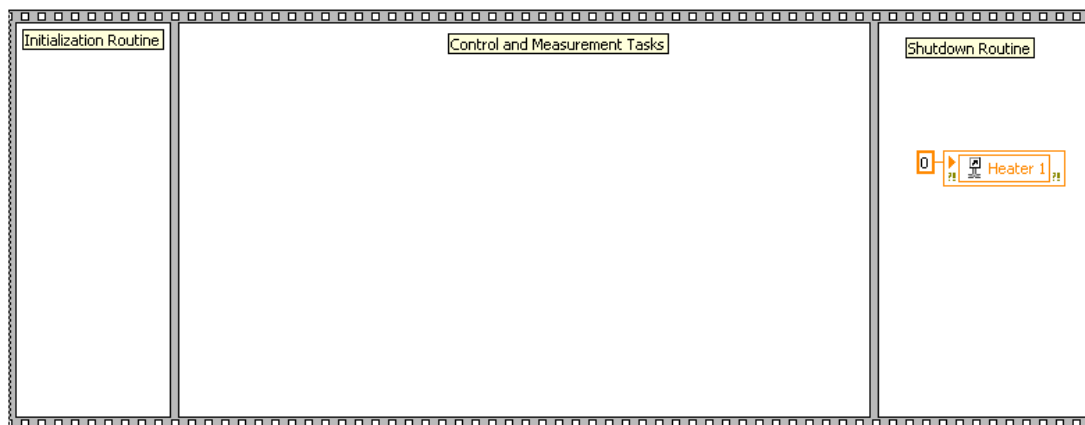


图2.5添加任务初始化或关闭逻辑

现在程序已有了一个完整的初始化和关闭规则，现在可以增加控制和测量任务了。

I/O 扫描和内存表



在LabVIEW8.6环境中，有一个属于Compact RIO的选项叫做RIO Scan Interface(RIO扫描接口)。当我们在Lab VIEW Project (Lab VIEW项目) 中发现Compact RIO控制器时，有选项选择控制器使用Scan Interface或Lab VIEW FPGA Interface (如果没有安装Lab VIEW FPGA，Lab VIEW默认为Scan Interface)。

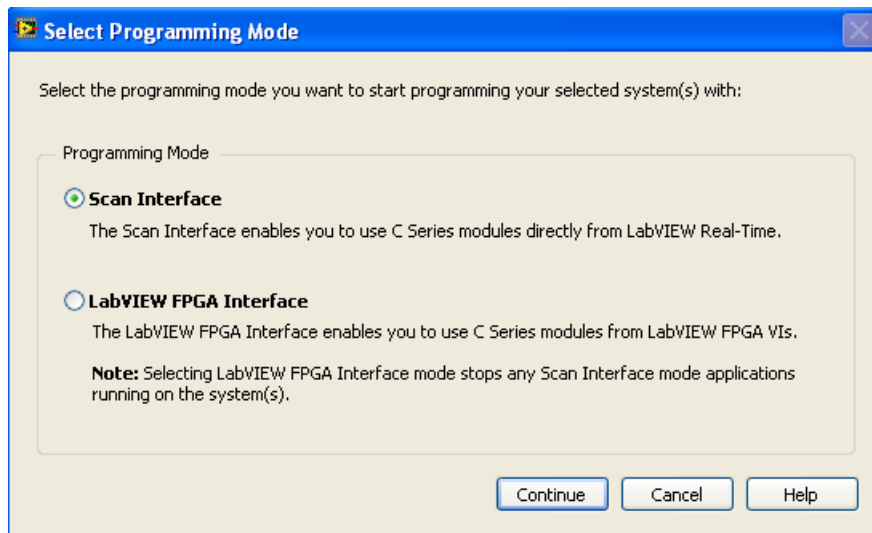


图2.6在LabVIEW8.6中，使用Scan Interface来编制Compact RIO控制程序

当控制器通过Scan Interface操作I/O时，自动从模块中读取I/O并且放置在Compact RIO控制器的内存表中。I/O scan的默认速率是10ms，可以在控制器属性中配置。使用I/O变量别名可以操作I/O。

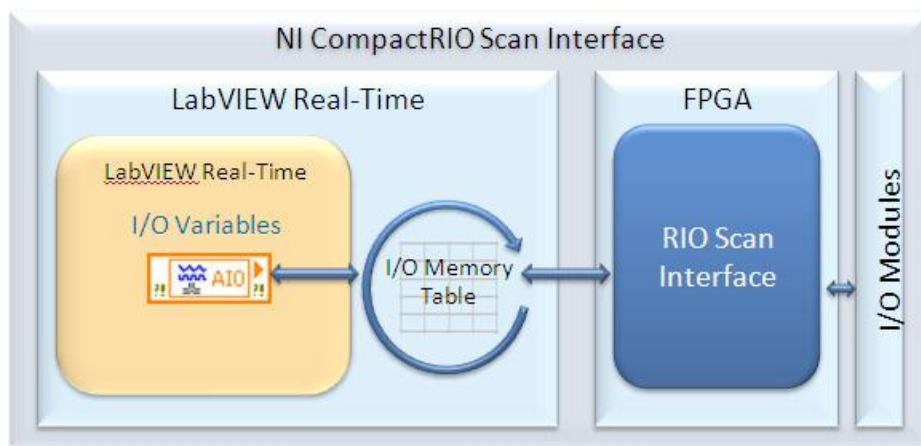


图2.7,描述Compact RIO Scan Interface软件单元的框图

在案例的系统中，有一个热电偶输入模块和一个PWM输出模块。我们可以通过Scan Interface配置和操作这两个模块。为了从LabVIEW中读写这些值，给这些项创建别名。一个I/O别名代表物理I/O。

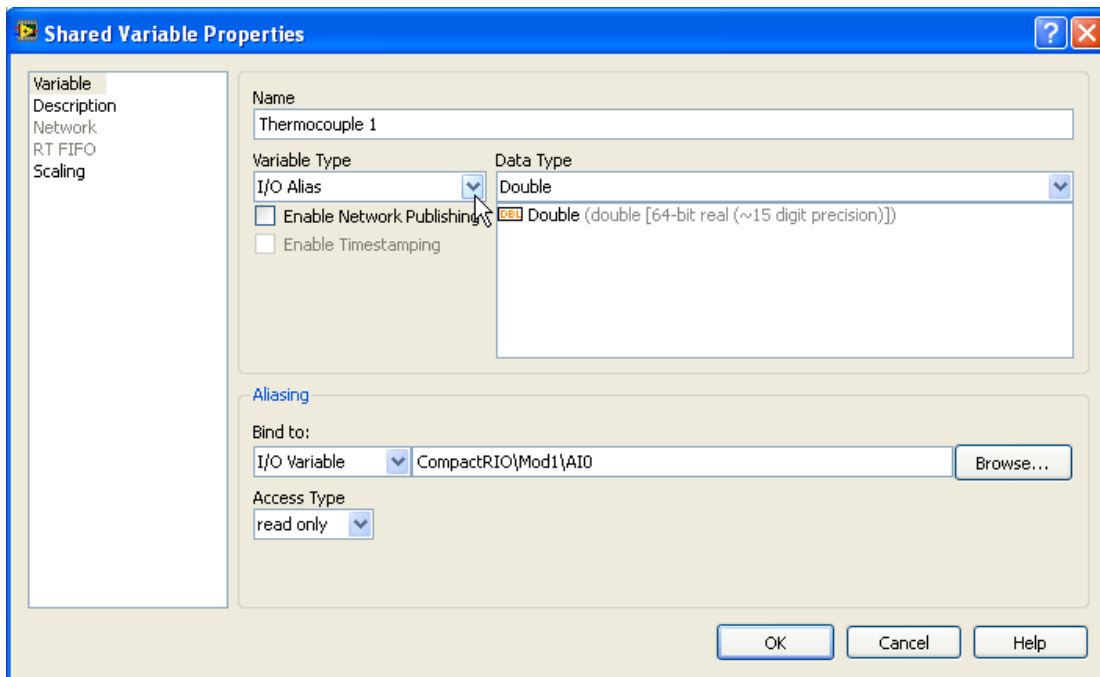


图2.8创建一个I/O别名

- 1 创建I/O别名，右键单击控制器选择一个新变量。选择变量类型为I/O Alias并将它与物理I/O 绑定。
- 2 在这里案例中，分别为Thermocouple 1(与一个TC模块绑定) 和Heat 1(为PWM输出配置的数字输出模式) 创建两个I/O别名，并将这两个I/O别名放入I/O library。

控制和测量任务

Process Control

使用定时循环安排每个控制和测量任务。程序应同步定时循环和I/O 扫描 (NI 扫描)，从而正确的同步化控制任务和I/O。

1. 创建一个定时循环并且配置其与扫描同步。设置周期(period)为1，表示循环执行一次I/O 扫描执行一次。

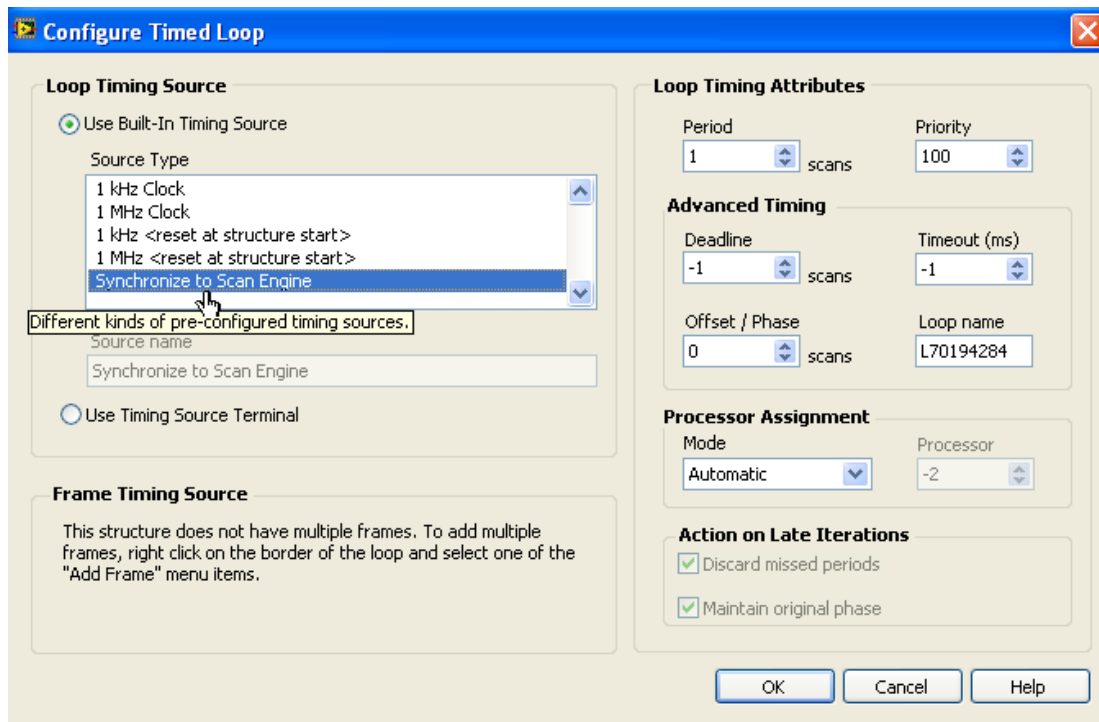


图2.9同步NI扫描引擎

2. 编写控制逻辑，包括：从I/O别名中读取输出、运行逻辑、写入I/O别名。

通常，我们编写子VI使代码能重用；由于本案例目的是演示整个架构，程序代码比较初级。在后面的案例中会讲解更多的代码封装。在这个简单的案例中，在程序面板拖入一个PID VI，连线常量使输出范围为[100, 0]，PID增益为[10, 0.1, 0]，Setpoint为350。若不使用常量而使用变量，我们可以在程序运行时重配置参数。连线“Temperature 1” I/O别名至“Process Variable”终端，连线“Heater 1” I/O别名至“Output”终端。创建正确的错误处理单元。

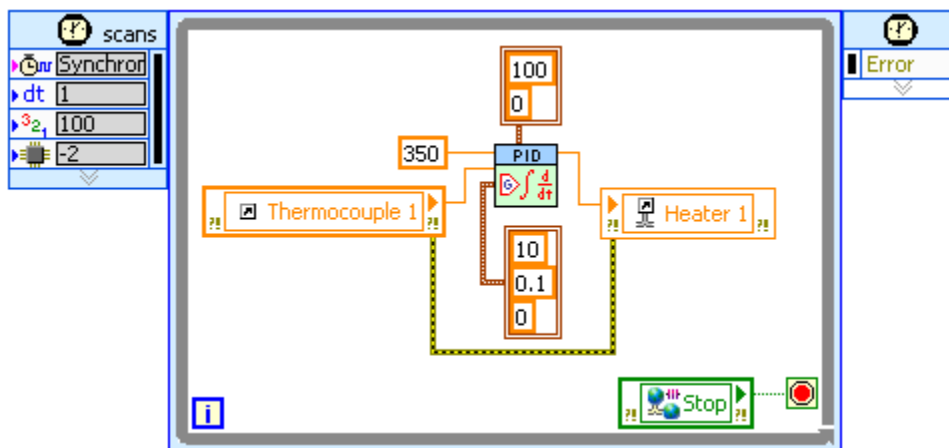


图2.10使用网络发布的共享变量来停止循环

现在我们可以运行这个温度控制程序了。程序有启动和关闭过程、Scanning架构、可重用的子VI以及错误处理机制。这就是整本指南中用于机器控制的基础架构。

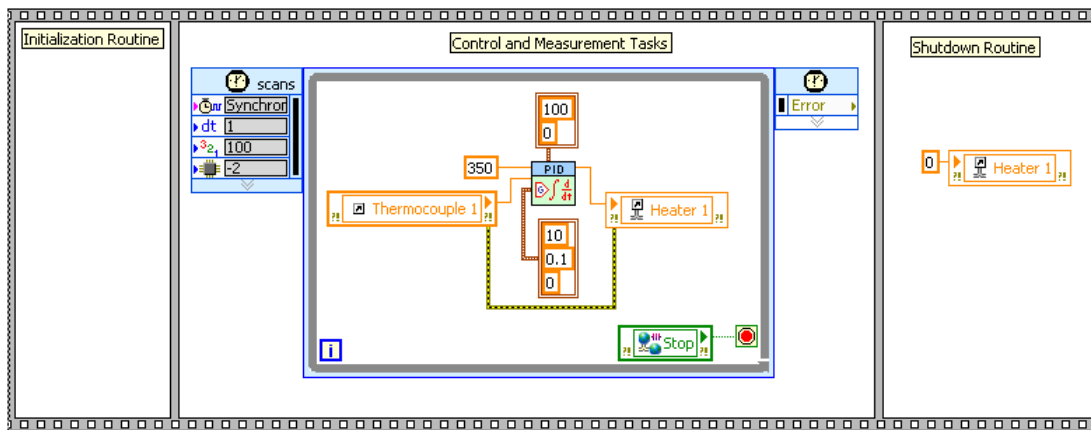


图2.11机器控制的基本构架

基于状态的程序设计

使用这种基础的架构，我们可以建立复杂机器控制的应用。但是，当逻辑比较复杂的时候，需要使用正确的逻辑架构来完成程序设计。通过建立软件架构，我们可以创建具有可扩展性、易维护的应用程序。使用由一系列状态构成的架构系统是设计可扩展、易维护的软件代码的普遍方法。

状态机概述

状态机是一种普遍而有效的软件架构。我们可以利用状态机设计模式来实现状态图或流程图的一些算法。状态机通常阐述一个适当复杂的决策算法，如诊断方法或过程监测。

状态机包括一系列状态和映射下一个状态的转换函数。当每个状态机处于某个状态或出口时，状态机通过入口来执行动作。因为状态机属于一个较大的机器控制架构的一部分，不能使用等待声明或循环，除非保留状态或执行算法，如用于数组处理的for循环。

在状态显著的应用软件程序中使用状态机。若我们能够将一个应用软件程序分解为几个不同的运行区域，状态机将是一个很好的软件架构。每个状态能够引导进入另一个或多个状态或者结束工序流程。状态机依赖于用户输入或状态内计算来确定进入下一步的状态。很多应用软件程序需要一个初始化状态，其后是一个缺省状态，在缺省状态中我们可以执行一系列动作。和状态一样，这些动作也依赖于先前和当前的输入。通常使用一个关闭状态来执行清理操作。

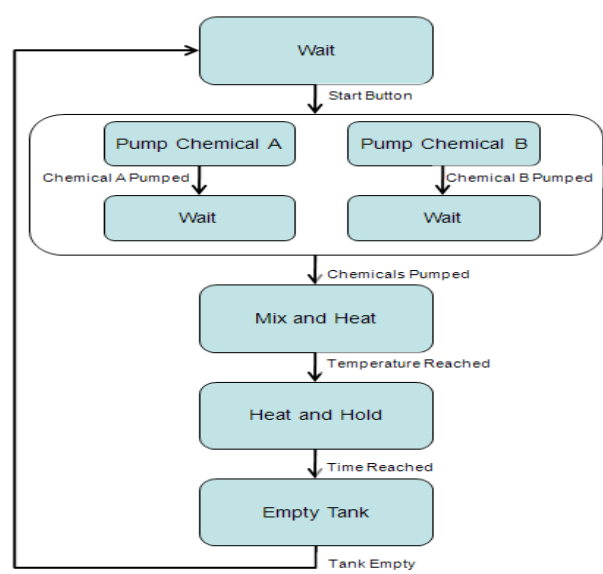
使用状态机的例程

为了解使用状态机架构如何给应用软件程序带来众多好处，我们设计一个用于化学反应容器的控制系统。在此应用软件程序中，控制器需要做到：

1. 等待操作员通过按钮发出指令；
2. 测量两个化学流体流速(两个并行过程)；
3. 在充满容器后，运行搅拌器并升高容器内温度。当温度达到200F时，关闭搅拌器并保持10秒温度恒定；
4. 将容器内液体泵入存储罐内；
5. 返回等待状态。

Lab VIEW中的状态机例程

在此程序中首先映射逻辑和I/O点。由于程序中包含一序列步骤，用流程图比较好规划该软件程序。以下是该应用软件程序的流程图和I/O信号表。



I/O Signals	I/O Name
Operator push button	Input_Operator_PB
Pump A	Output_PumpA
Pump B	Output_PumpB
Chemical A Flow	Input_ChemA_Flow
Chemical B Flow	Input_ChemB_Flow
Stirrer	Output_Stirrer
Heater	Output_Heater
Thermocouple	Input_TC
Drain Pump	Ouput_PumpDrain
Tank Empty Level Sensor	Input_TankEmpty_LS

图2.12程序流程图及I/O信号列表

状态机的每个状态执行唯一的动作以及调用其他状态。依靠是否有顺序或条件的发生实现状态的转换。将状态转换框图转化为Lab VIEW的程序框图需要以下的组成部分：

- 条件结构- 包含每个状态的条件和执行代码
- 寄存器 – 包含状态转换信息
- 状态的函数代码 – 实现每个状态的功能
- 转换代码 – 在顺序上决定下一个状态

一旦我们在系统中定义了状态，就可以在Lab VIEW中使用条件结构来为每个状态表示和容纳逻辑操作。

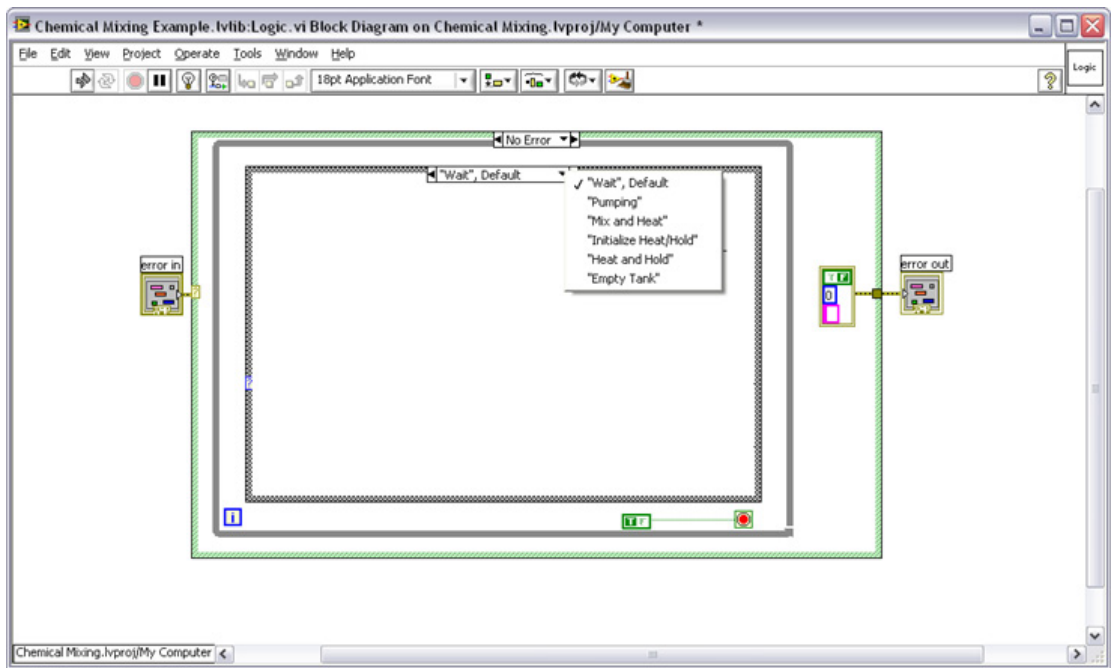
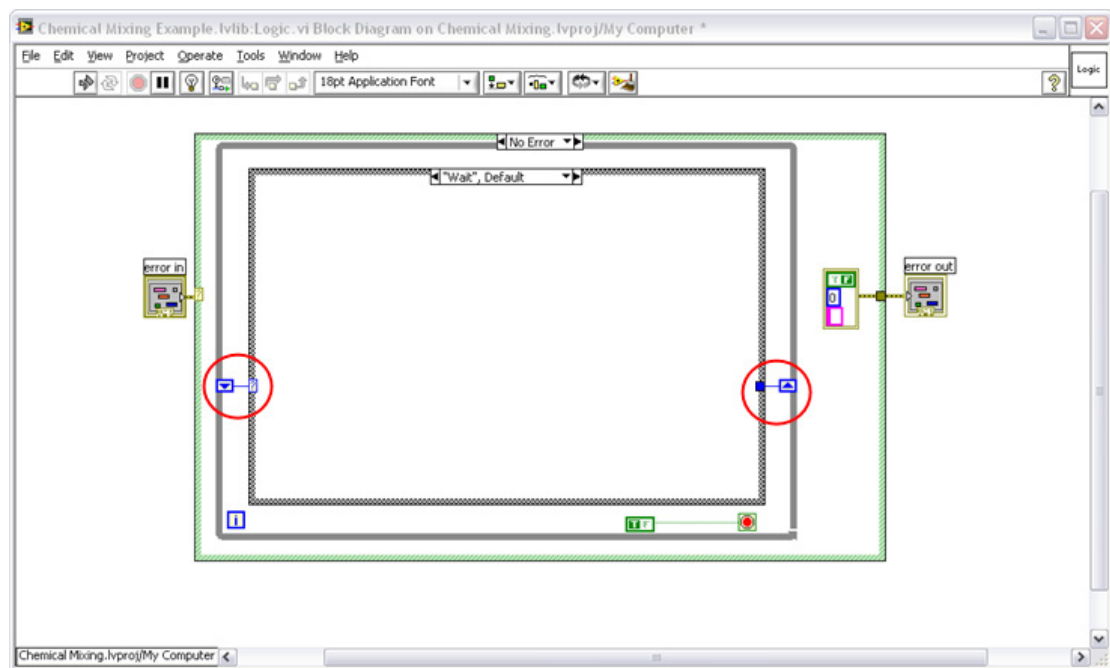


图2.13为每一个状态创建一个选择结构

通过在应用程序中添加寄存器，在每次状态执行的时候，我们可以跟踪或传递当前的状态信息。条件结构的输入终端与寄存器连接。



2.14添加移位寄存器来传递数据

现在，在条件结构中的每个状态可以写入一些每次都需执行的Lab VIEW代码。图2.15的代码中为混合和加热化学反应容器执行了一个PID函数。

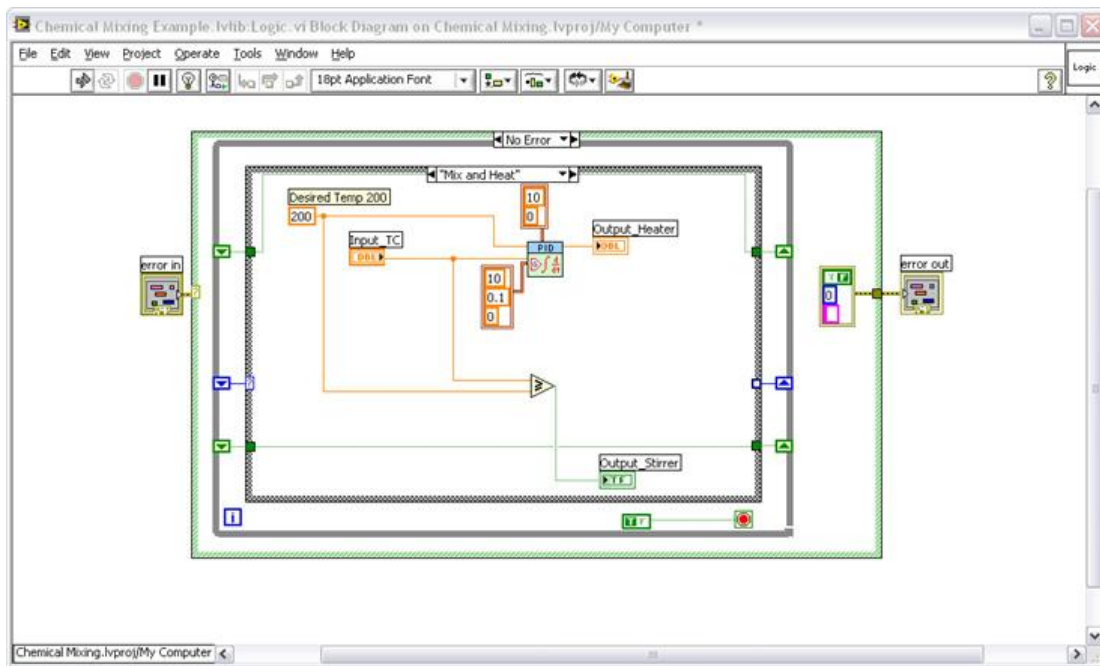


图2.15在每个状态中写入程序代码

通过系统中的条件，每个状态必须确定下一个状态。通过添加转换逻辑确定下一步执行的状态。

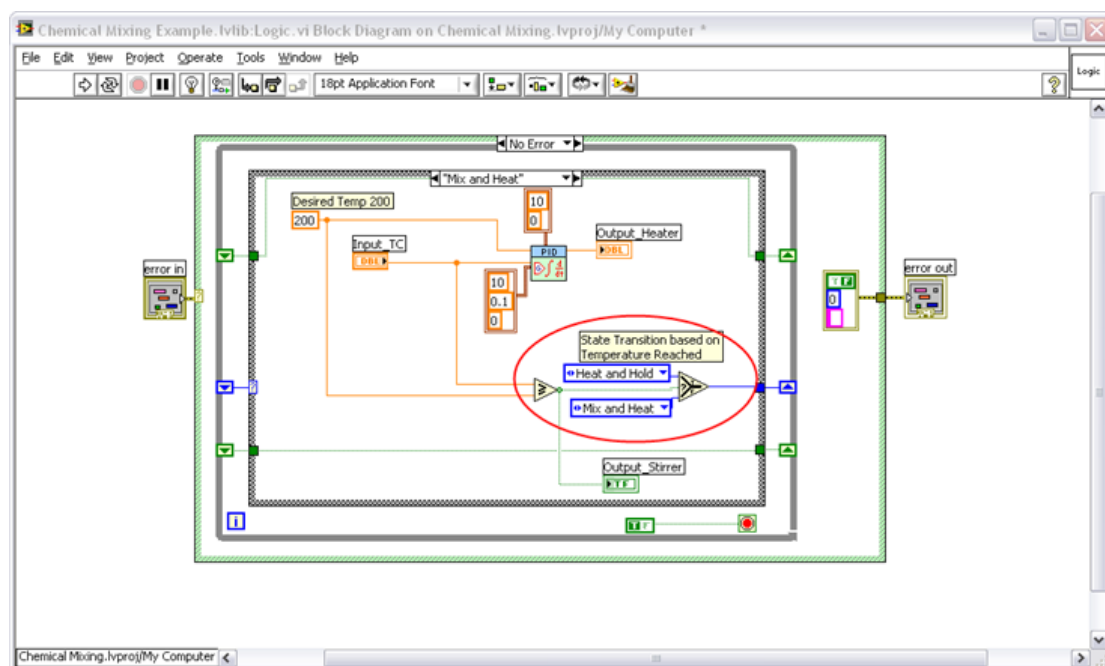


图2.16使用一个选择器来决定下一个状态

在这个例程中，为了测试逻辑操作，我们使用了模拟I/O代替物理I/O。使用一个全局变量代替硬件读写VI。在应用于真实硬件之前，通过交互式的输入控件和显示控件能够非常便利的测试程序的逻辑操作。

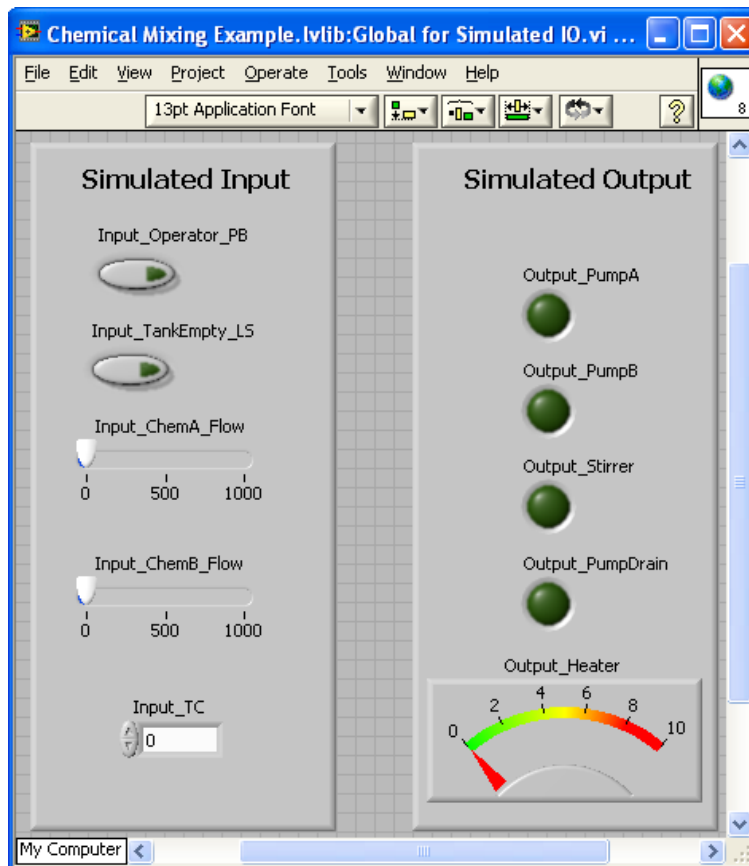


图2.17不用硬件，只使用全局变量可以很方便的来测试程序

因为状态机的每个状态里都具有并行的状态，所以需要建立第二个状态来匹配上个状态里的并行逻辑状态。只有都两个状态都计算完毕才能退出。

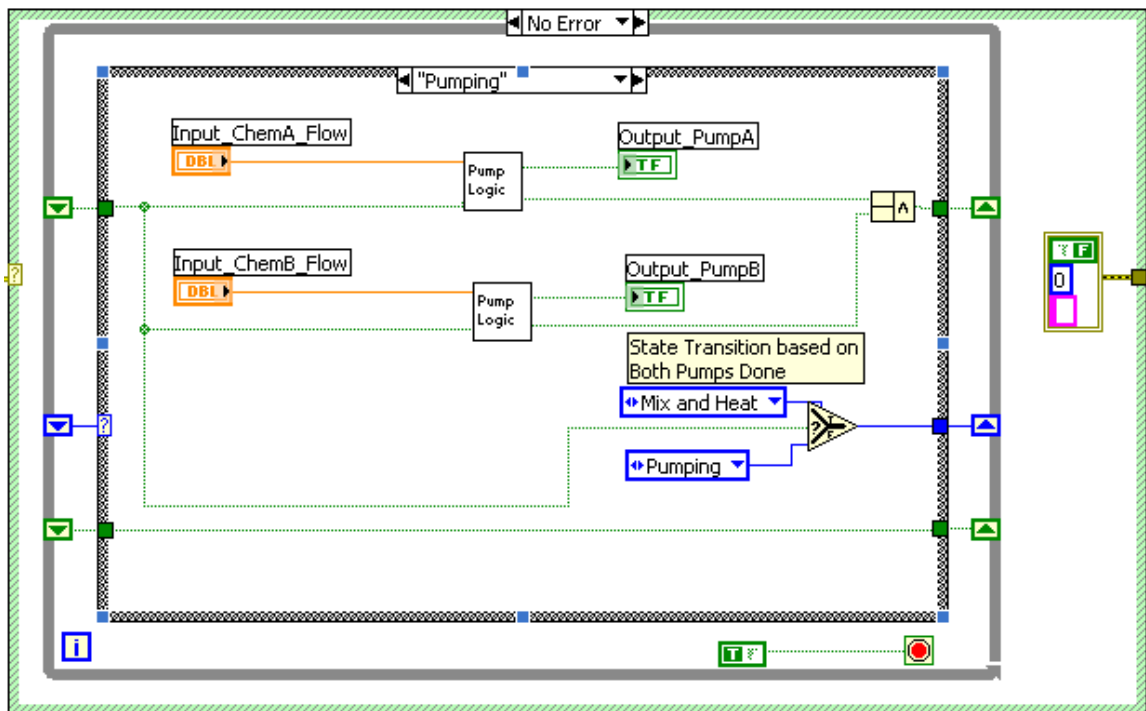


图2.18一个状态可以并行处理多个任务

状态图表介绍

要理解状态图(statechart)，最好先了解经典状态图(state diagram)，然后再了解嵌套、并发、事件等概念。经典状态图由两个主要结构组成：状态和状态转移。图2 中的经典状态图描述了一个简单的饮料贩卖机，其中有5 个状态和7 个描述状态机运行方式的状态转移。机器从“空闲”状态开始，当投入硬币后，将转移到“硬币计数”状态。该经典状态图中还显示了贩卖机等待用户选择、送出饮料和找零这三个阶段的状态和转移。

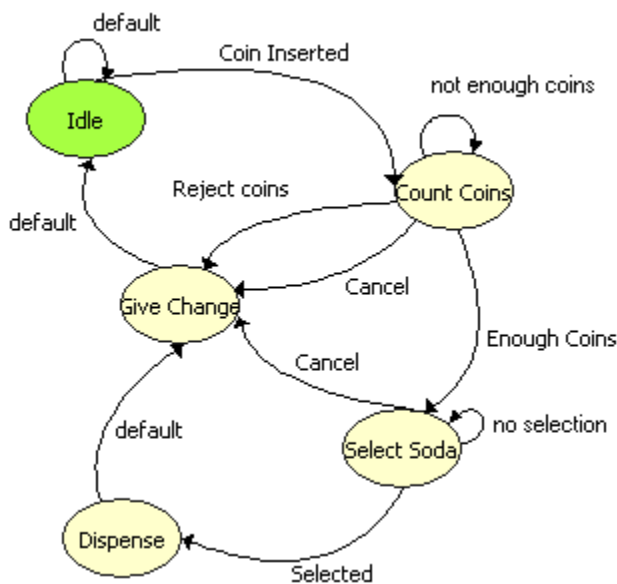


图2.19一个饮料自动贩卖机的状态图

图3 中的状态图描述了同一个饮料贩卖机的行为。请注意嵌套和事件怎样实现了状态和状态转移数目的减少。在状态图中，可以将“硬币计数”和“送出饮料”这两个状态组合在一个超状态中。你只需要在这两个状态中的任一状态和“找零”状态之间定义一个转移(T3)。T3 状态转移可以响应3个事件：饮料送出、请求找零或硬币弹出。另外，在经典状态图中，可以在状态转移T2 中引入一个“警戒”条件，以省去“选择饮料”状态。要触发转移，警戒条件必须为true。如果警戒条件为false，则事件将被忽略，不触发转移。

Vending Machine Statechart

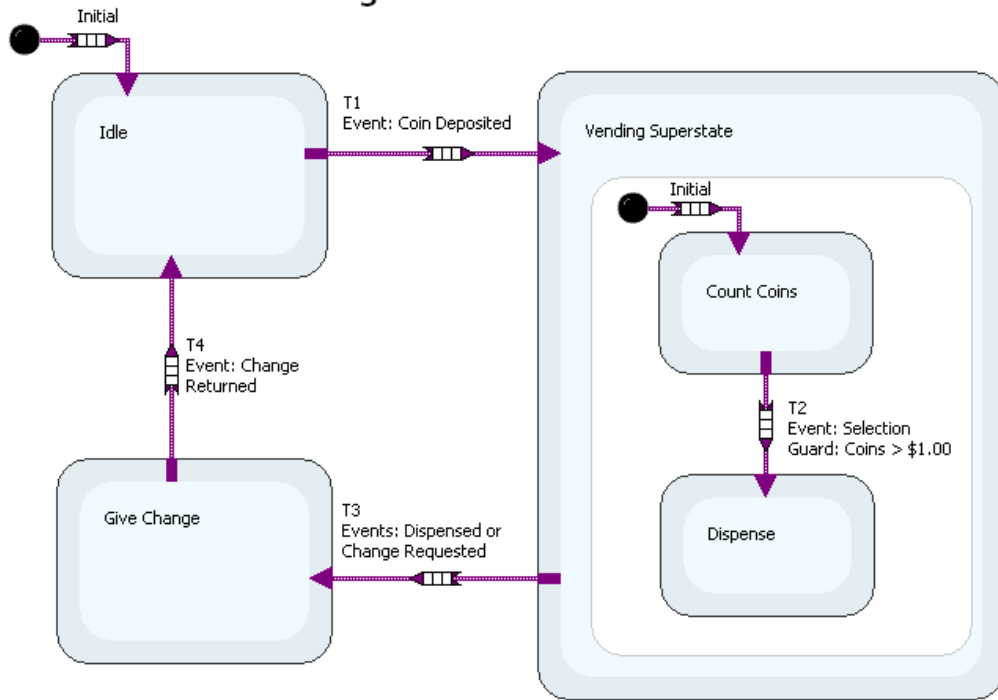


图2. 20一个简单的饮料自动贩卖机的状态图

这时，我们可以通过在贩卖机的软件中增加一个温度控制元件，来扩展该状态图，并说明并发的概念。图2.20中显示了如何将饮料贩卖逻辑与温度控制逻辑封装到一个与状态中。与状态所描述的系统能在同一时间处于两个彼此独立的状态中。**T7** 转移显示了状态图怎样定义两个子状态图的退出动作。

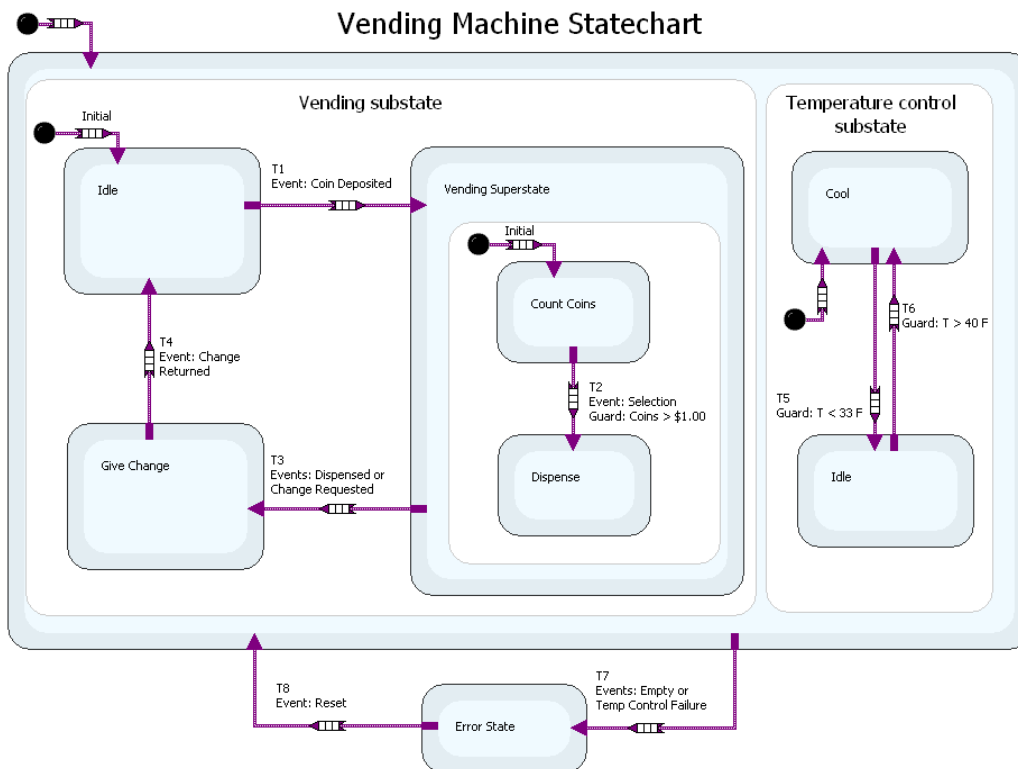


图2. 21 T7转化显示了一个状态表怎样定义应用到两个子状态表的退出功能

除了嵌套和并发外，状态图的其他一些特点对复杂系统的设计来说也非常有用。状态图中的“历史”允许一个超状态来“记录”它上一次的激活子状态。例如，假设某个超状态描述了一种机器，该机器在注入某种物质后对其加热。在机器注入物质的时候，暂停事件会暂停机器的注入操作；当恢复事件发生时，机器则会继续执行刚才的注入操作。

状态图

状态图由域(region)、状态(state)、伪状态(pseudostate)、转换(transition)和连接器(connector)组成。

域

域是指包含状态的区域。顶层状态图是一个包含了所有状态的域。另外，你还可以在某个状态中创建域：即利用层次式设计的方法，在某个状态的内部创建其他状态。下图中描述了这种层次式设计功能：在一个状态的内部，通过域创建了一个子状态。每个域中都必须包含一个初始伪状态。

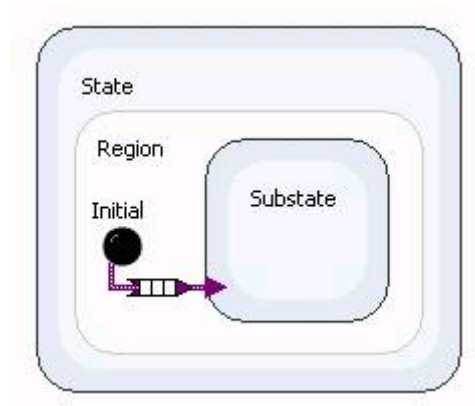


图2.22 在一个状态里使用域来创建一个子状态

状态

状态是指状态图所能存在的某个阶段。状态必须位于域中，而且至少拥有一个进入的转换。

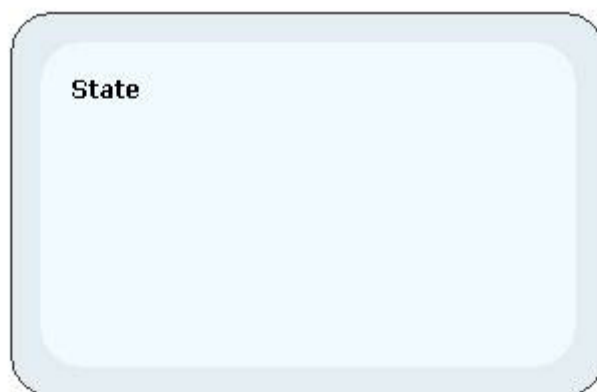


图2.23 状态就是状态表的条件

每个状态都有一个相关的**进入**和**退出动作**。进入动作是指进入某个状态时所执行的Lab VIEW 代码。**退出动作**是指离开某个状态时(在转换到下一个状态之前)所执行的Lab VIEW 代码。每个状态都只能有一个进入和退出动作，而且这两个都是可选的。每次进入或退出某个状态时，都会执行进入与/或退出动作。

可以通过Configure State 对话框来访问该代码。

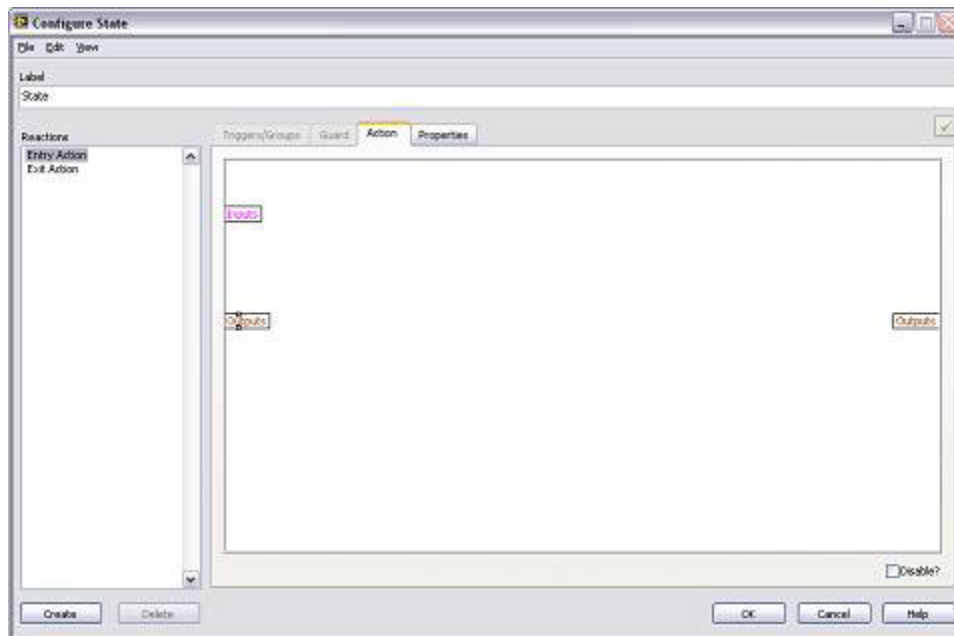


图2. 24通过配置状态对话框就可以访问进入和退出程序代码

可以进一步对状态进行配置，使其具有**静态反应**。静态反应是指状态没有执行任何进入或转出转换时所执行的动作。一个状态可以有多个静态反应，状态图的每次迭代中可能会执行这些静态反应。

每个静态反应都由三个部分组成 - 触发器、警戒条件和动作。

触发器是指触发状态图执行的事件或信号。异步状态图只有接收到触发器后才会执行 - 例如，按钮或其它用户界面交互可以产生一个触发器。触发器的值传递到状态图中，然后状态图基于触发器再执行相应动作。在同步状态图中，触发器则周期地自动传递到状态图中。触发器的默认值是NULL。

监护条件是指在执行状态动作之前所测试的一段代码。如果监护条件为真，则将执行动作代码；如果监护条件为假，则不执行。如果状态图接收到一个触发器（该触发器将由某个特定的静态反应来处理）并且监护条件的值为真，则将执行该动作代码。

动作是指完成预期状态逻辑的Lab VIEW 代码，可以是输入，内部状态信息的读取以及相应的输出更改。你可以通过Configure State 对话框，新建一个反应动作来创建这种静态反应。一旦新建了一个**反应动作**，你就可以将它与触发器相关联并设计监护条件和动作代码。只有静态反应才能配置有触发器和监护条件。

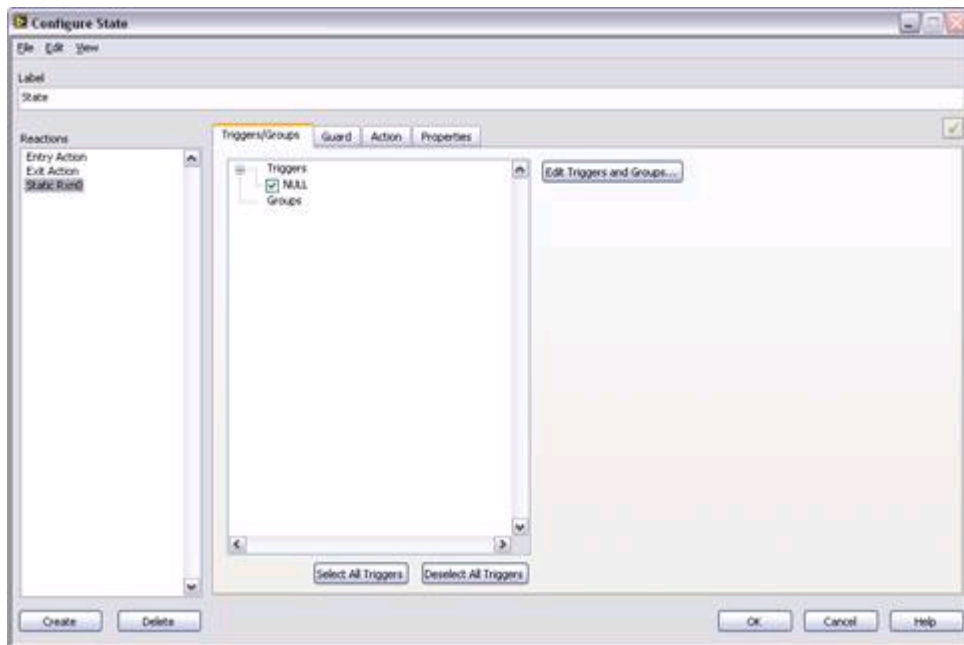


图2.25通过配置状态对话框来创建一个新的状态反应

正交域和并发

当状态具有两个或两个以上的域时，这些域就称为是正交的。下图中的域1 和域2 就是正交的。

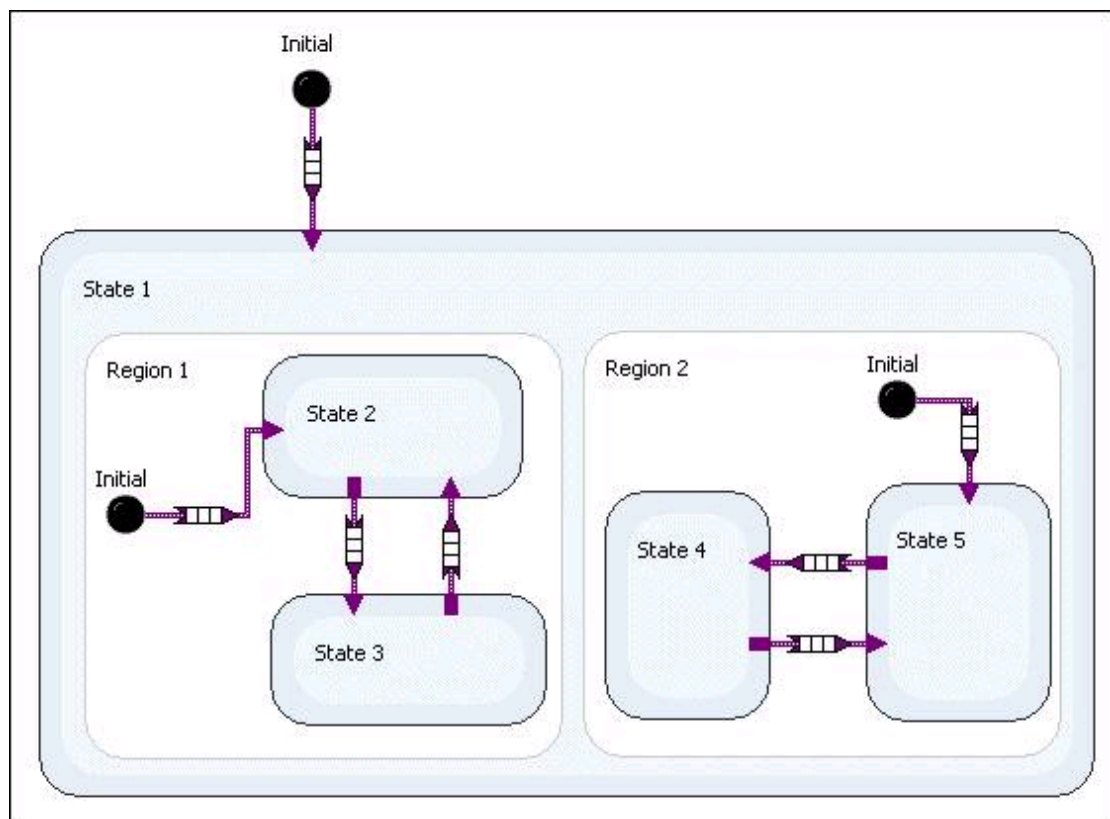


图2.26域1和域2正交

正交域中的子状态是并发的，也就是说当超状态都处于激活状态时，状态图在每次迭代中都可以进入每个正交域中的某个子状态。并发与并行是不一样的。在状态图的每次迭代中，并发的子状态是轮流被激活的，而并行子状态则是同时被激活的。Lab VIEW 状态图

模块不支持并行的状态激活。

转换的描述

转换定义了状态图在两个状态之间的转换条件。

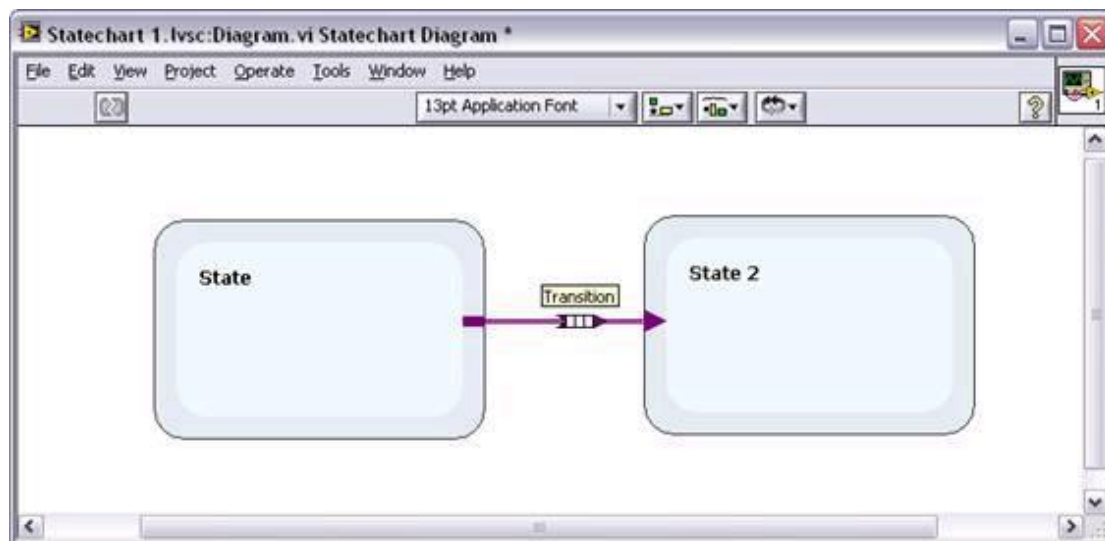


图2. 27转换定义了来给你个状态间转换的条件

转换由**端口**和**转换节点**构成。端口是指状态之间的连接点，而转换节点则基于触发器、监护条件和动作定义了转换的行为。用户可以通过**Configure Transition** 对话框来配置转换节点。

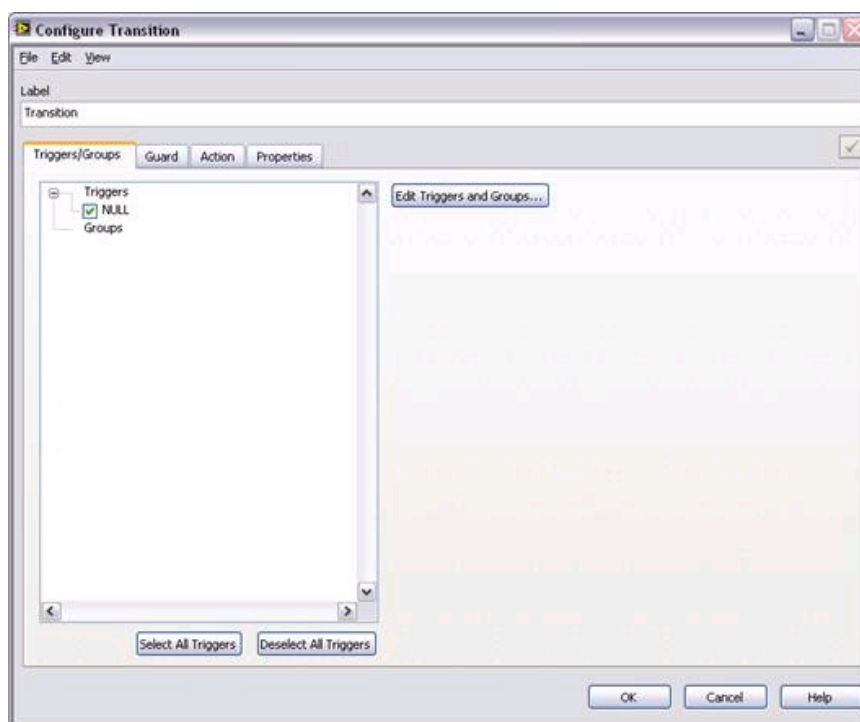


图2. 28通过配置转换对话框来配置转换节点

转换中的**触发器**、**监护条件**和**动作**同状态中定义的触发器、监护条件和动作是一样的。触发器会触发转换的发生；如果监护条件为真，则将执行动作，而状态图也会转换到下一个状态。如果监护条件不为真，则不会执行动作代码，状态图也不会转到转换所指向的下一个状态。

伪状态

伪状态是一种状态图对象，表示一种状态。Lab VIEW 状态图模块中包括以下几种伪状态：

- 开始状态 - 是指进入域时首先出现的状态。每个域中都必须有一个开始状态。
- 终止状态 - 是指域中的最后一个状态，结束域中所有状态操作。
- 浅度历史 - 当状态图离开域然后再返回时，状态图重新进入在它退出域之前的最高一级的活动子状态。
- 深度历史 - 当状态图离开域然后再返回时，状态图重新进入在它退出域之前的最低一级的活动子状态。

连接器

连接器是一种状态图对象，将多个转换片段连接起来。Lab VIEW 状态图模块包含以下几种连接器：

- 叉形 - 将一个转换片段分开成多个片段
- 合并 - 将多个转换片段合并到一个片段
- 连接 - 将多个转换片段连接起来

Lab VIEW中的状态图表例程

为证明Lab VIEW Statechart模块的优点，我们沿用了前面使用状态机的例子：

1. 为了解使用状态机架构如何给应用软件程序带来众多好处，我们设计一个用于化学反应容器的控制系统。在此应用软件程序中，控制器需要做到：
2. 等待操作员通过按钮发出指令；
3. 测量两个化学流体流速(两个并行过程)；
4. 在充满容器后，运行搅拌器并升高容器内温度。当温度达到200F时，关闭搅拌器并保持10秒温度恒定；
5. 将容器内液体泵入存储罐内；
6. 返回等待状态。

首先为每个I/O信号创建I/O别名的library。

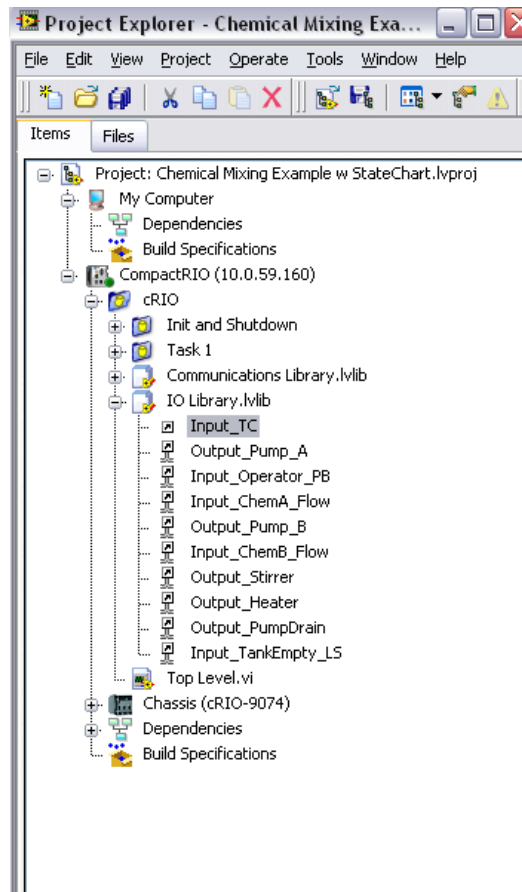


图2. 29为每个I/O信号创建一个I/O别名库

接下来创建关闭任务来为输出I/O别名设置默认输出状态。

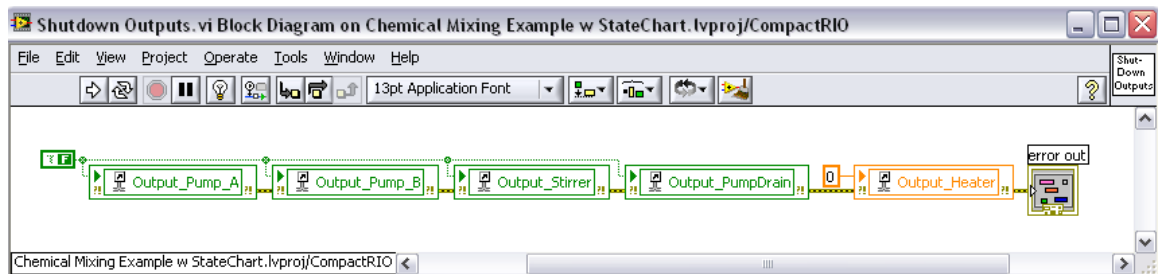


图2. 20创建一个关闭任务来设置输出I/O别名的默认输出状态

开发基于状态图表的应用程序的过程包含以下步骤：

1. 设计Caller VI
2. 定义输入、输出和触发器
3. 开发状态图表
4. 将状态图表放入Caller VI

第一步 设计调用 VI

这个应用程序中的顶层VI是Caller VI。它是连续调用状态图表的一个定时循环。此外，这个顶层VI还包括了开始和关闭的代码，这些代码封装在一个顺序结构中。

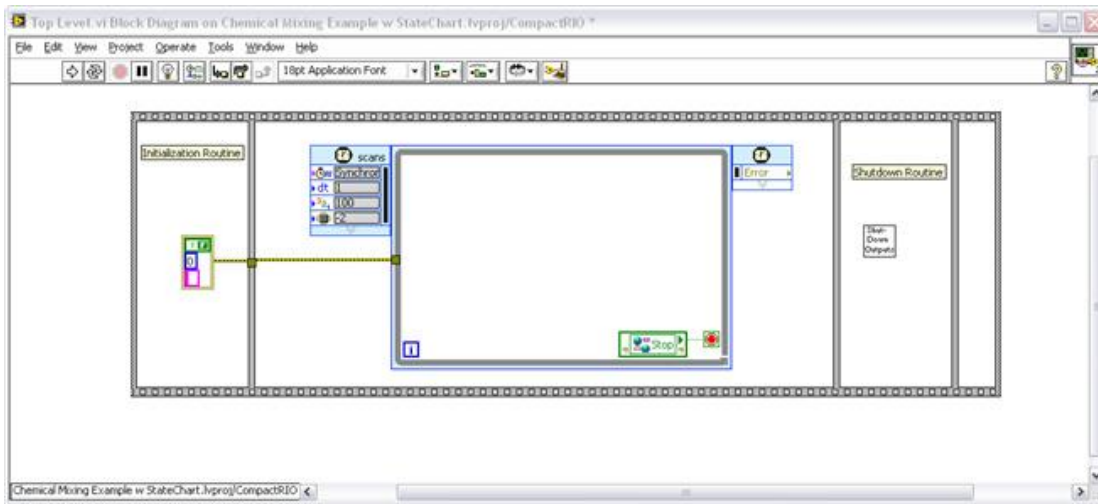


图2.31 顶层VI包含连续调用状态表的定时循环、开机以及关机的程序代码，这程序代码放置在一个顺序结构里
现在在Lab VIEW项目中添加一个新的状态图表。每个状态图表都包含有几个组成部分，用以配置所要设计的内容。

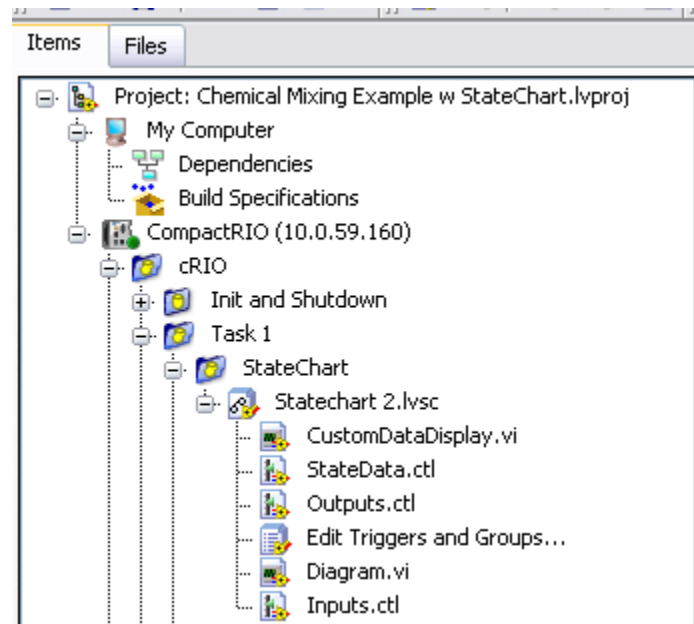


图2.32向Lab VIEW项目里添加一个新的状态表

Diagram.vi文件内有实际的状态图表。Inputs.cti和outputs.cti是定义状态图表中输入输出的簇。Statedata.cti是仅在状态图表中使用的内部状态信息。在这个例子中，不使用触发器、statedata.cti和customdatadisplay.vi。

第二步：定义输入、输出以及触发

打开、修改并保存输入类型定义（inputs.cti）和输出类型定义（outputs.cti），来为每个I/O点创建一个输入和输出。输出类型定

义里包含一个错误簇，此错误簇用来显示状态表抛出的错误条件。

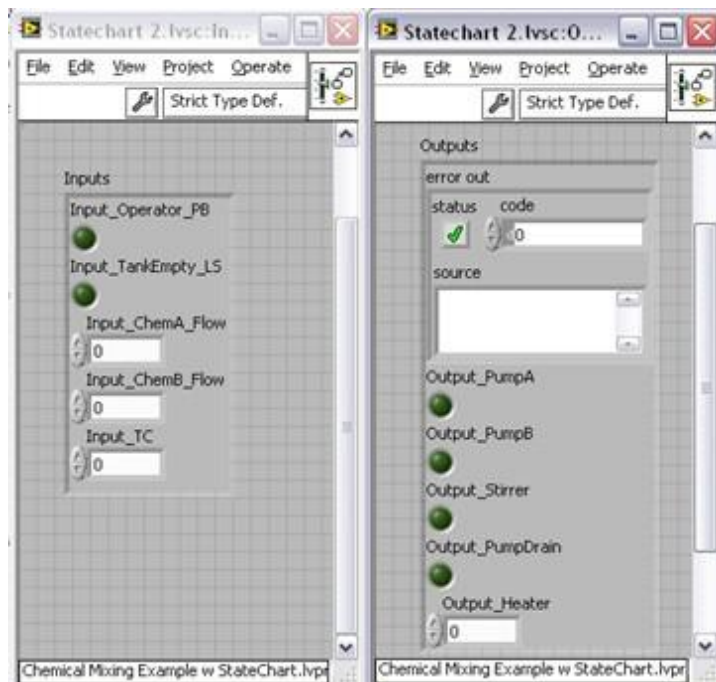


图2.33 打开、修改并保存输入类型定义（inputs.cti）和输出类型定义（outputs.cti），来为每个I/O点创建一个输入和输出

第三步：创建一个状态图

打开diagram.vi文件。在程序框图里，创建系统的状态以及状态间的转化。创建适当的状态、范围以及转化来表示编程逻辑。每个状态及转化都包含了Lab VIEW代码，当被激活时，这些代码开始执行。状态表是一个异步状态表，它使用触发来决定什么时候开始进行状态转化。使用状态表的一个主要优点就是可以直观地表示系统的行为。

第四步：将状态图放置到调用VI里

编写完程序之后，立即点击左上角的图标来生成状态图的程序代码。

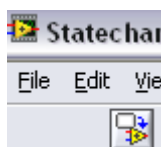


图2.34点击左上角的图标来生成状态图的程序代码

在主要应用程序里，将状态表拖放到代码的逻辑部分。因为状态表需要使用簇来进行输入和输出，所以需要创建子VI来读取I/O别名，并将他们传入状态表的簇里或从簇里传递出去。输出子VI在向变量写入数据前会检查错误条件。如果发生错误，输出子VI就会将错误直接传递出去而不用写入变量。

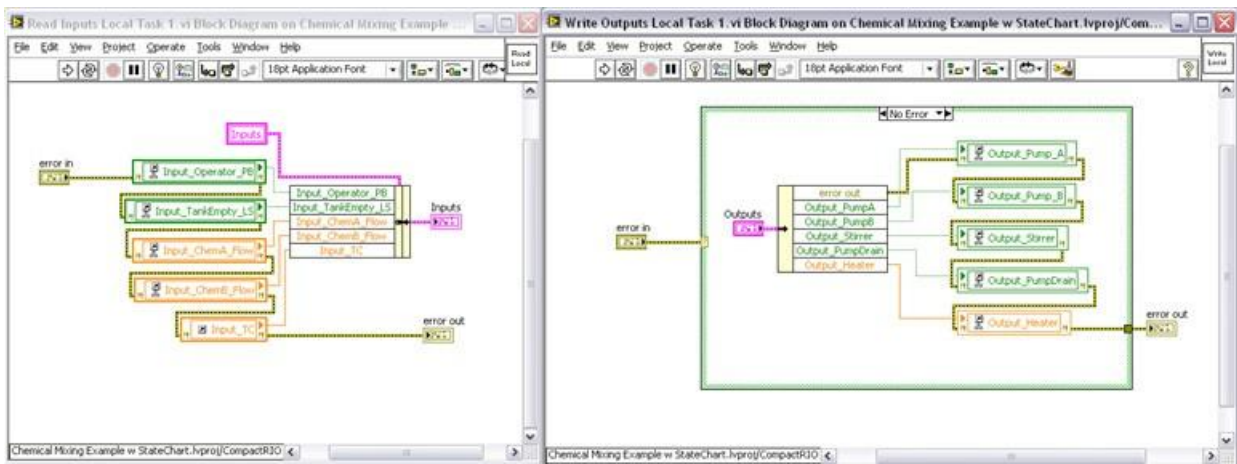


图2.35创建子VI来读取I/O别名，并将他们传入状态表的簇里或从簇里传递出去

最后，将所有的函数都放置或连线到主VI。将状态表放置在一个条件结构里来检查是否有错误发生。如果发生错误，那么程序就会跳过状态表来执行。这样就能对程序的错误状态进行可靠的检查，从而保证控制系统能够正常的执行。右击状态表进入属性，将状态表设置为可以调试。这样通过使用Lab VIEW的高亮执行以及调试工具比如断点、指针以及单步执行，就可以对状态表进行直观地调试。在配置为最佳性能之前一定禁止调试。

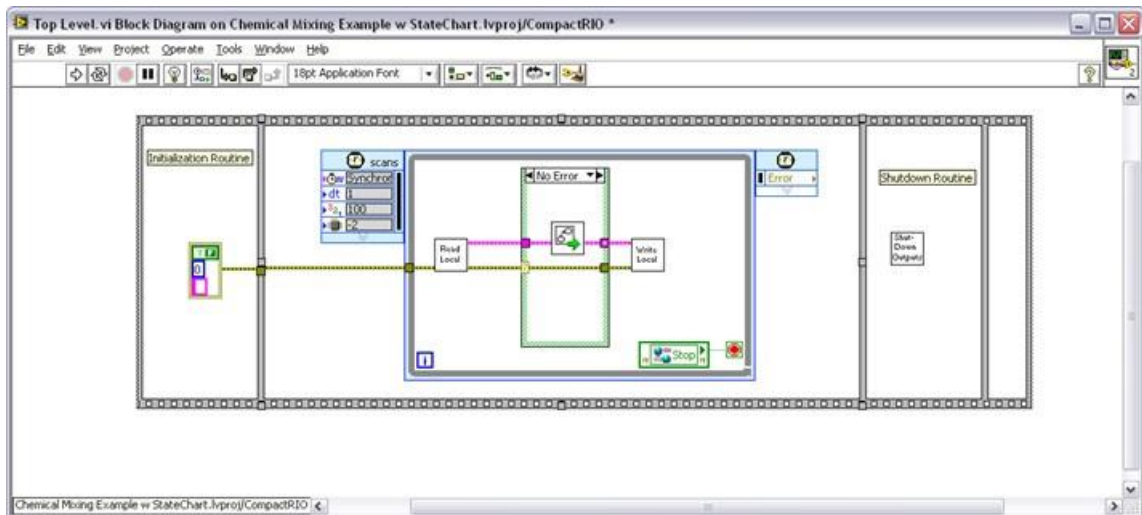


图2.36如果没有错误，那么状态表就一直执行

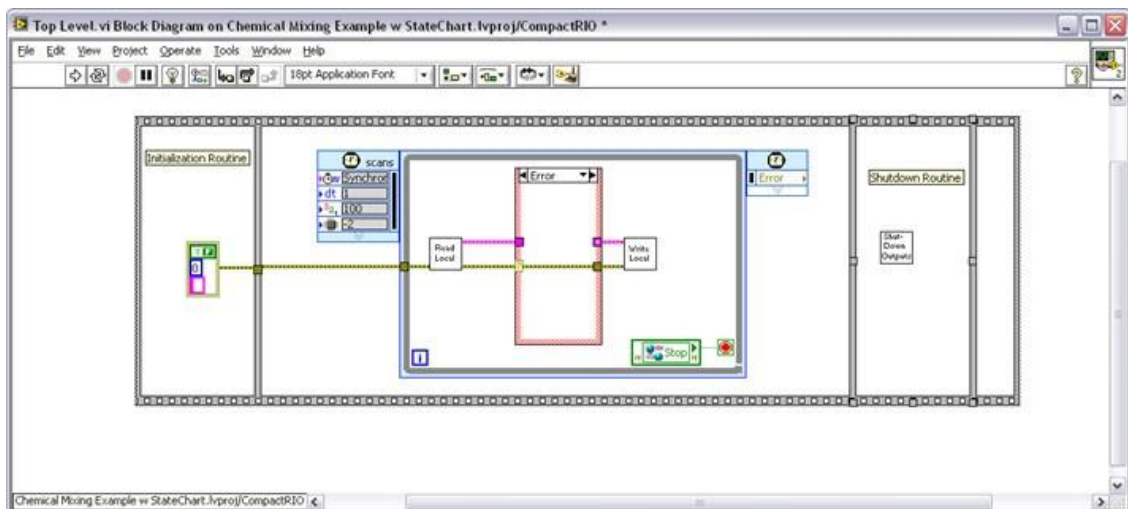


图2.37如果发生错误，状态表就停止执行

开始执行—修改例子

修改现存的实例来实现上面的设计方法。在这节中，将修改先前创建的化学混合例子，使用状态表来创建自己的应用程序。主要有四步：

1. 修改I/O库，来为应用程序的物理I/O创建IOV别名
2. 修改关闭程序，来为物理输出写入关闭值
3. 修改任务1，来从状态表中读取或向其写入I/O
4. 修改或重新编写适合应用程序的状态表

第一步：修改I/O库

打开 **Chemical Mixing.lvproj**。因为应用程序使用不同的I/O，所以需要修改I/O库，来为物理I/O创建IOV别名。如果已经完成了连线，就可以将IOV别名映射到物理I/O了。如果没有完成连线，也可以将来再映射IOV别名。扩展项目里的I/O库。可以通过双击别名来同时编辑变量。最快的方法就是使用**Multiple Variable Editor**。通过双击I/O库并选择“**Multiple Variable Editor...**”来打开编辑器。

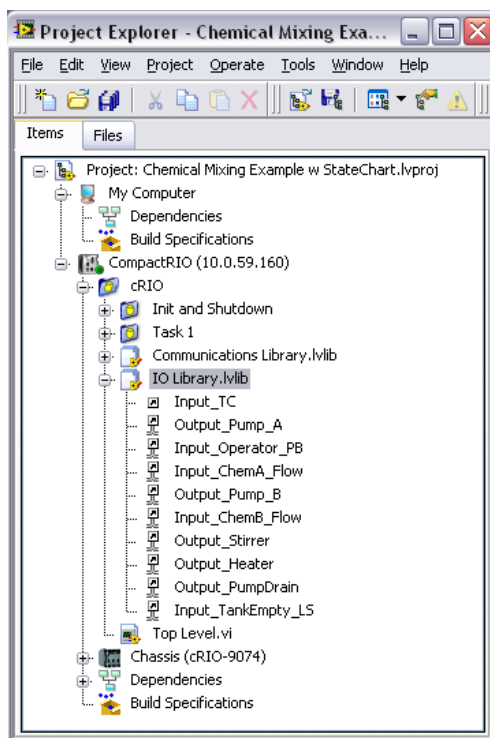


图2.38使用Multiple Variable Editor来快速编辑变量

1. 在**Multiple Variable Editor**里修改变量的名称、数据类型和物理绑定。也可以通过复制、粘贴已经存在的变量来快速创建新的变量。

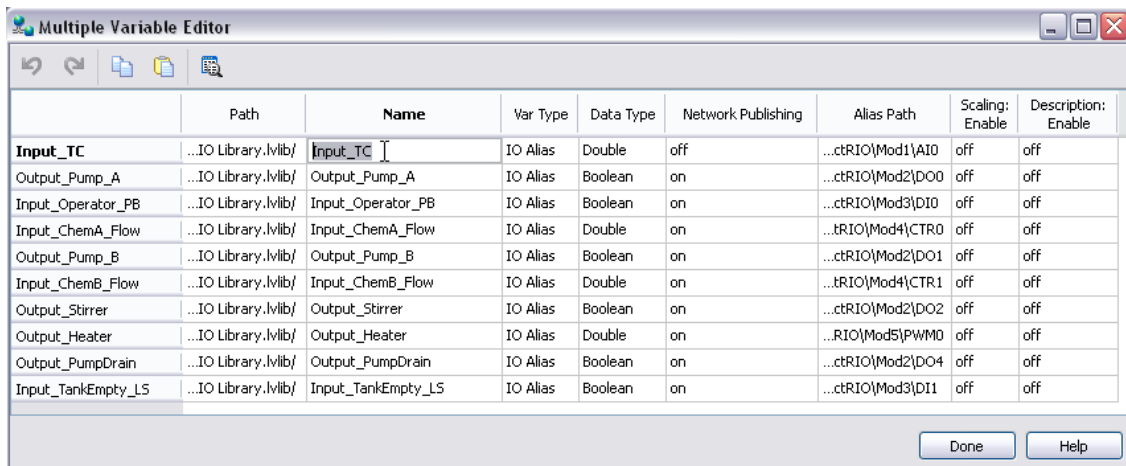


图2.39 Multiple Variable Editor选项

第二步：修改关闭程序

因为应用程序使用不同的I/O，所以需要修改关闭程序，来设置输出的关闭值。

- 打开Shutdown Output.vi，并修改此函数，为新建的IOV别名设置默认输出值

第三步：修改任务1来影射I/O

接下来需要修改逻辑。因为每个逻辑任务都为程序的执行创建一个局部的I/O副本，所以需要重新影射I/O。

在状态表文件夹里，打开输出类型定义（outputs.cti）和输入类型定义（inputs.cti）文件。修改这些类型定义来匹配应用程序的I/O。

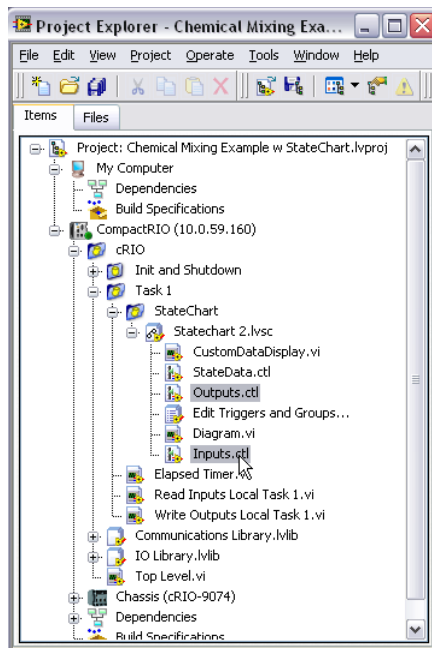


图2.40打开输出类型定义（outputs.cti）和输入类型定义（inputs.cti）文件并修改这些类型定义来匹配应用程序的

- 更新Write Outputs Local Task1.vi和Read Outputs Local Task1.vi来读写I/O。

第四步：修改或重新编写状态表

接下来就可以输入逻辑。通过修改或重新编写状态表来执行应用程序就可以实现逻辑的输入。

第三章

开发可扩展性系统的软件技巧

重用函数

当设计机器控制代码时，最好将部分代码设为可重用。这样可以节省你的开发时间，因为你可以在一个项目里模块化代码并且建立一个代码库以备将来项目的使用。在其他开发环境中，这些可重用的代码块被称为函数或功能块。可重用的代码需要三个基本条件：

1. 必须有一个方法来调用程序并提供输入输出数据；
2. 为保持状态，代码必须保持它自己的内存空间（别的函数可以不需要此条件）
3. 在一个程序中，代码必须具备有多个实例的功能。

创建Lab VIEW里的可重用代码

在Lab VIEW中，可重用的代码块称为子VI。Lab VIEW是一种层次语言，通过可重入的子VI能很容易的实现代码的可重用。可重用代码同样遵循三个基本条件：

1. 必须有一个方法来调用程序并提供输入输出数据。
 - 在Lab VIEW的前面板上创建输入输出，并将这些控件连线到连接板，这样就能实现这个方法。

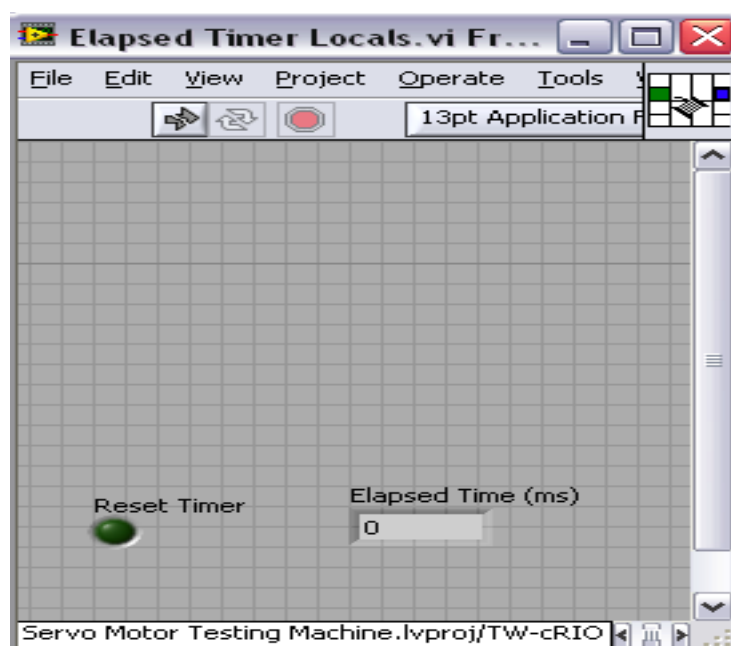


图3.1在连接板上将输入和输出连接到前面板的控制和显示控件上

2. 为保持状态，代码必须保持它自己的内存空间（别的函数可以不需要此条件）。
 - 在Lab VIEW中，可以用两种方式实现。可以使用带有未初始化的移位寄存器的While循环或者创建局部变量。局部变量的系统开销会略大一些但较容易理解。你可以从前面板的控制和显示控件创建局部变量。在控件上单击右键创建局部变量。这些变量可以在程序面板里多次被引

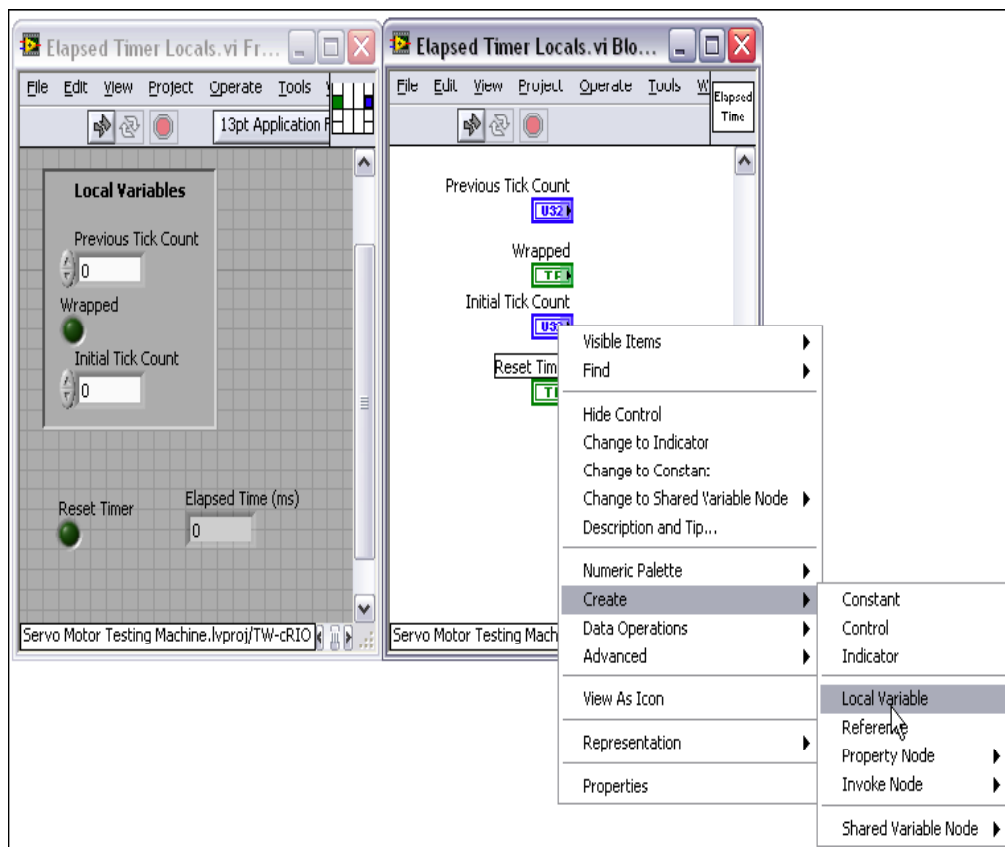


图3.2 右击控制或者显示控件来创建局部变量

3. 在一个程序中，代码必须具备有多个实例的功能。

- 在Lab VIEW中，通过设置VI可重来实现。程序中可重入的VI为每个被调用的实例建立独立的内存空间。在File下的VI属性页里，在Category下拉菜单中选择Execution，选择Reentrant execution。

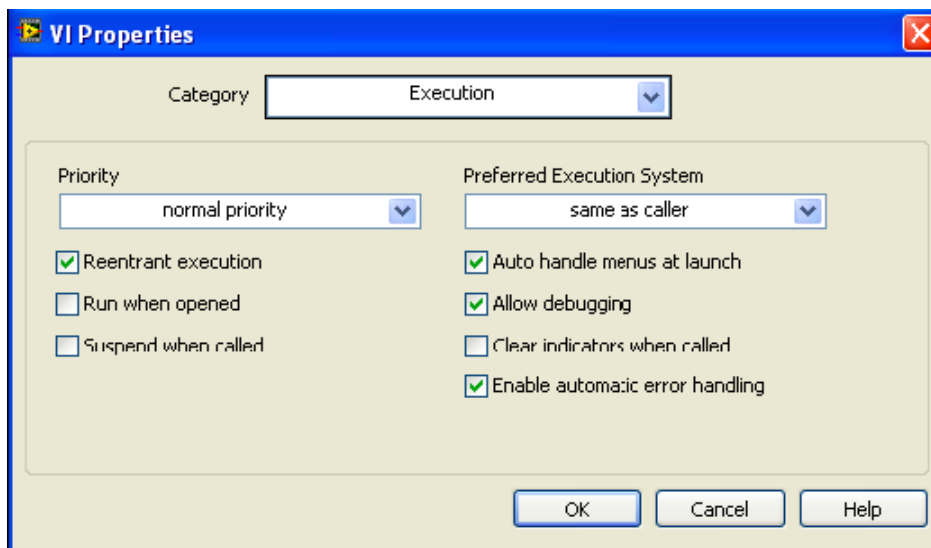


图3.3 使VI可重用

创建可重用代码例程

在前面化学混合的例子中，在一个特定的时间段内，需要将混合物保持在一个设定的温度。由于控制应用程序必须保持响应，所以不能使用“wait”声明去控制应用程序的定时。如果你使用了“wait”声明，当程序处于等待状态时，控制算法的其余部分不会运行。这样所拥有的程序是一个无响应的应用程序。由于不能在循环中加入“wait”声明，你需要一种方法，可以在每次迭代中，检查已运行的时间。这是一个普遍的需求而且是可重用代码的理想应用程序。

为了解如何建立一般的可重用代码，创建一个子VI来确定已运行的时间。该函数应输出已运行时间并有重设定定时器的输入。在Lab VIEW中，Tick Count函数是一个达微秒量级的计时器。Tick Count函数输出一个U32类型的微秒值。以下是计算已运行时间的子VI的逻辑：

- 检查是否该VI是第一次运行或重置计数器输入是否为“真”。若是“真”，读取Tick Count并存储为初始的计数时间，设已运行时间输出为0，并且在相关的寄存器写入“假”。
- 与上次Tick Count输出值比较，检查Tick Count输出值是否有变化（若输出值超过U32的可利用空间，则从0开始）。若Tick Count有变化，设置相关的寄存器内的值为“真”。
- 从初始的Tick Count减去现在的Tick Count。如果计数已结束，那么将计时时间转化为U64，加上 $2^{32}-1$ 之后，再减去初始的计时时间，然后再将结果转化为U32

1. 必须有一个方法来调用程序并提供输入输出数据：

- 建立一个新的VI。在前面板为“reset”和“elapsed time”分别创建控制和显示控件。将它们与连接器面板连接。

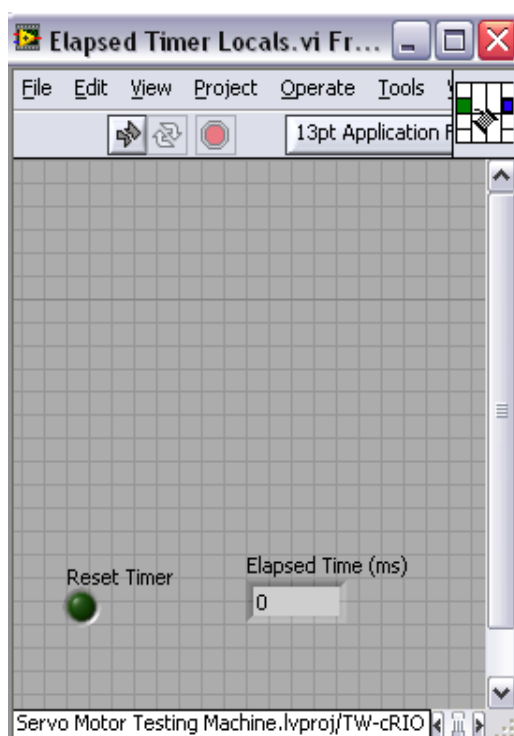


图3.4 创建新VI来调用程序并提供输入和输出数据

2. 为保持状态，代码必须保持它自己的内存空间（别的函数可以不需要此条件）

- 在前面板为三个局部变量创建控制和显示控件（先前计时时间、结束时间、初始计时时间）。

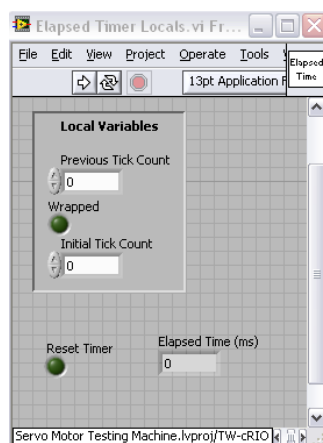


图3.5为三个局部变量创建控制和显示控件（先前计时时间、结束时间、初始计时时间）

3. 在一个程序中，代码必须具备有多个实例的功能。

- 进入属性窗口设置VI为可重入

现在，在程序面板上为计时器编写逻辑代码。当你需要存取局部数据时，右键单击控制和显示控件并创建局部变量。现在，可以调试代码并可以在多个控制程序中重用该VI了。

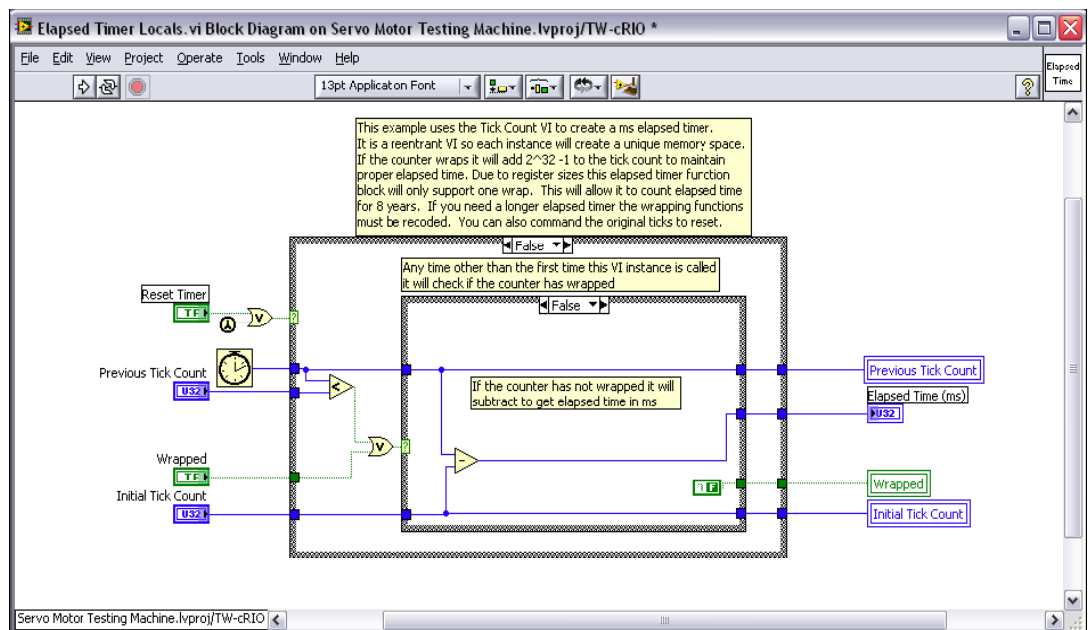


图3.6 完成的程序

Lab VIEW里的其他可重用代码

NI的Lab VIEW自带一个拥有大量可重用代码的库，你可以通过Function面板使用这些可重用代码。这些可重用代码提供了几百个内置函数用于控制、分析、通信、文件I/O等等。

IEC 61131 功能块

LabVIEW8.6引入了一种新的可重用代码类型，称为功能块。这些功能块基于编写工业控制系统的IEC 61131-3国际标准。代码语言为Lab VIEW，为实时应用程序设计，具有在内存表（共享变量）中发布参数的能力。你可以与其它Lab VIEW代码一起使用这些功能块。

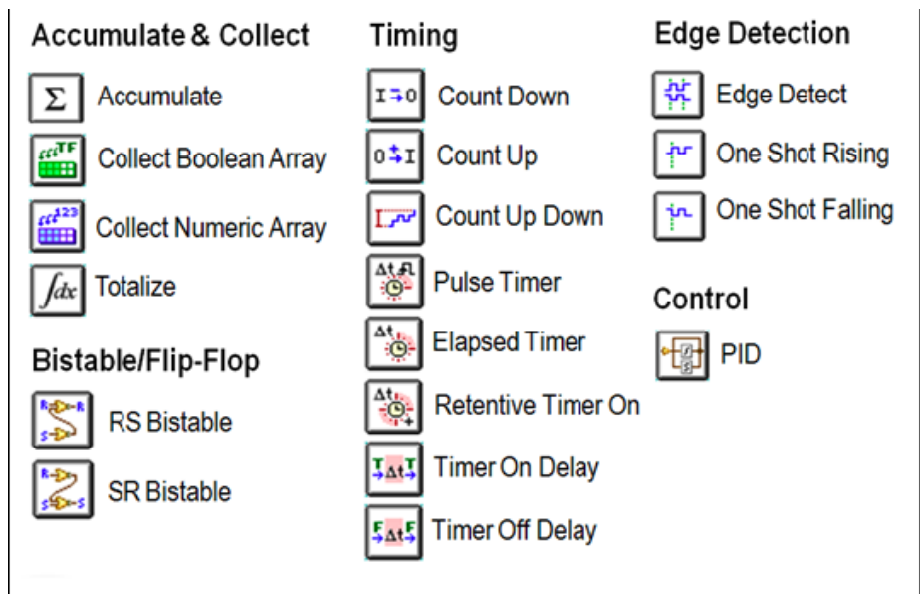


图3.7基于编写工业控制系统的IEC 61131-3国际标准最新Lab VIEW 功能块

可配置的终端变量

功能块与标准子VI的不同之处在于通过提供配置页和选项来直接连接输入输出与Lab VIEW Project里可见的全局内存表。也可以通过网络进入这些入口。可以从功能块的Property窗口中配置端子和变量。

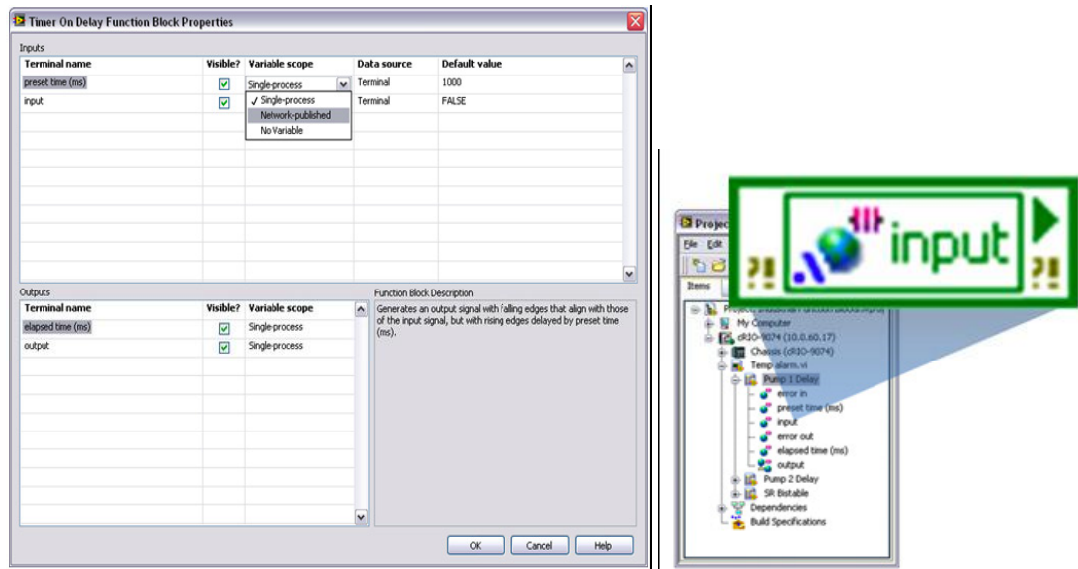


图3.8 配置属性窗口中的功能块以及从内存列表中访问输入输出

多任务（多循环）

在许多应用程序中，控制器需要运行多个控制和测量任务。举个例子，一个机器的控制应用程序可以有一个任务使用状态图表的方式控制机器运行，另一个任务执行机器健康监测或记录数据。控制程序里可以有多个任务并行运行并将递数据传递至内存表。

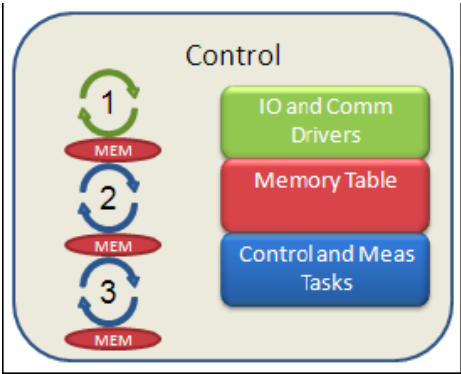


图3.9控制程序里可以有多个任务并行运行并将递数据传递至内存表。

为控制应用程序的执行，需要能够：

- 在任务之间设置优先权
- 同步任务
- 任务之间传递数据
- 触发任务

设置任务优先权及同步任务

当运行多个任务时，需要确定控制任务拥有最高优先权。因为Lab VIEW是基于时间运行的，高优先权的循环比如I/O Scan通常按照一个设定的时间表运行。这样能确保低抖动的运算和稳定的控制。但是，也意味着如果控制器没有按时完成被调用的运算，这些运算会被中断掉。对于像数据存储和网络通信这种低优先权的任务来说是可以的。但对于控制任务来说，一旦中断掉，将可能导致非稳定

运算。因此，应该为应用程序设计各个任务的优先权。你也可以使用工具如NI Real-Time Execution Trace Toolkit对你的应用程序执行基准测试，从而确保后台任务如通信有足够的时间。

可以使用定时循环来设置任务的优先权。相对于其它循环，定时循环具有可配置优先权的功能。你输入的数字越大，优先权越高。优先权数字必须是一个介于1和65535之间的正整数。Lab VIEW执行系统的特点是先占式的，所以高优先权的定时结构先于所有低优先权的结构执行，其它Lab VIEW代码在时序要求严格的优先权中不会运行。

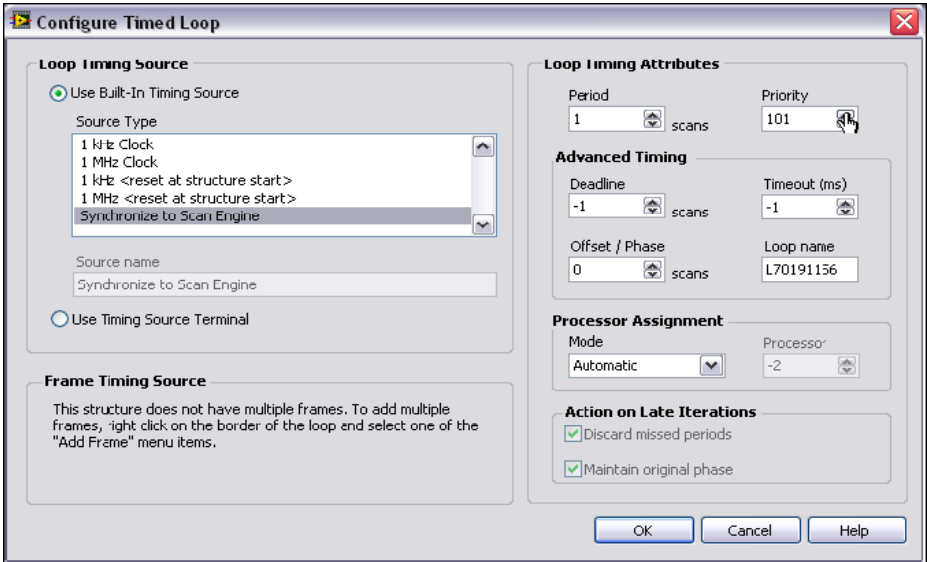


图3.10 设置定时循环的优先权

为了同步多任务，将它们设置为与NI 扫描引擎同步。所有与扫描引擎同步的循环以I/O 扫描速率运行，按优先权高低分别执行。若你有一些无需同步或优先执行的后台任务或非决定性任务，你可以在标准while循环中通过wait函数设置定时时间来运行这些任务。

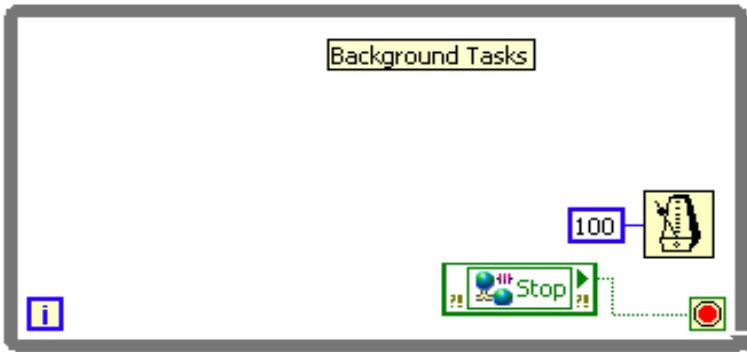


图3.11 普通优先权的While循环

在任务之间传递数据

所有任务都能够从内存表中读写I/O。为在任务之间传递数据，你需要在内存表中增加一组新的数据，这些数据叫做Lab VIEW共享变量。有了Lab VIEW共享变量，可以在控制器内或通过网络全局性地共享数据。它们是可配置的，通过控制器或网络，可以使用它们来提供功能。现在，重点在使用共享变量来向内存表中输入数据。

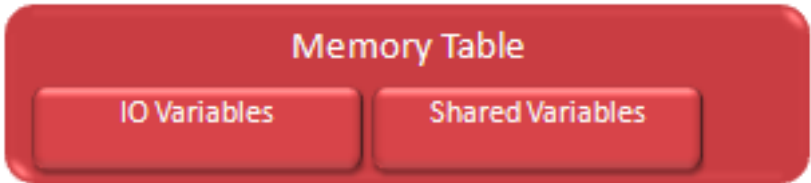


图3.12共享变量和I/O变量是内存列表的元素

创建新的共享变量与创建I/O别名类似。你可以通过使用库来组织这些变量。首先在项目中创建一个新的库然后创建一个新的变量。选择数据类型并设置变量类型为Single Process（单进程共享变量是全局变量）。

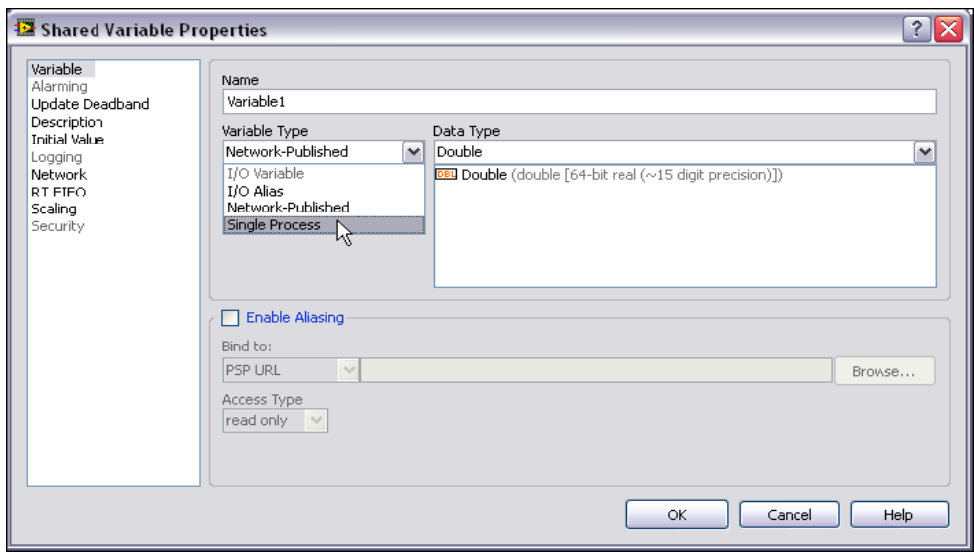


图3.13 创建共享变量

进入RT FIFO标签。一些变量如数组，在单处理器运算中是不能被读写的。当循环被高优先权循环先占时，未完的运算可能引起处理器使用率升高和抖动。为避免此现象，选择Enable RT FIFO并设置它为Single Element。

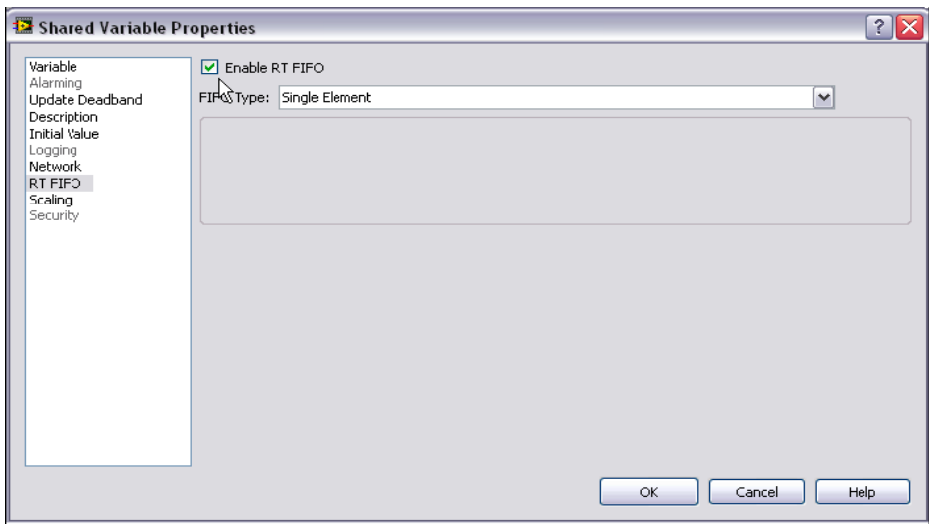


图3.14 通过使用实时的FIFO（先进先出），可以从多个循环中读取共享变量或者向其写入值，这不会引起抖动

现在可以从控制代码的任何地方向内存表读或者写了。

触发任务

有时需要从一个任务来触发另一个任务。例如，顶层机器控制代码可能控制整个机器，并且在某个状态需要开启波形采集和分析运算。波形采集运算需要在低优先权的并行任务中执行。为触发这些并行任务，需要使用一个基于命令的架构

多循环系统的基于命令架构

基于命令的架构可以简单描绘为具有两个实体的模型。这两个实体是指挥员和劳动者，两者通过消息和命令联系起来。指挥员通过消息传递要执行的命令给劳动者，而劳动者执行命令。

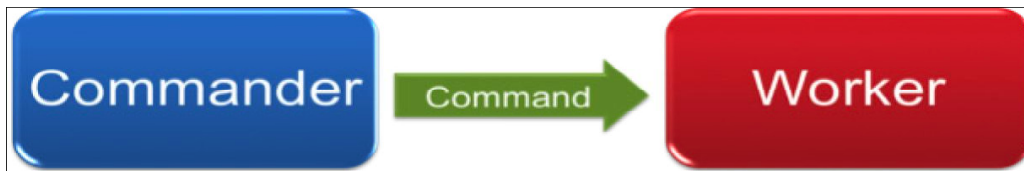


图3.15基于命令的结构提供了一个在循环任务中传递数据的机制

其它用于描述这个模型的术语，如主-从、服务器-客户端、生产者消费者模型等，在意义和实施方面差别不大。

在Lab VIEW中，有好几种方法可以在并行循环中传递命令，如实时FIFO、队列以及共享变量。

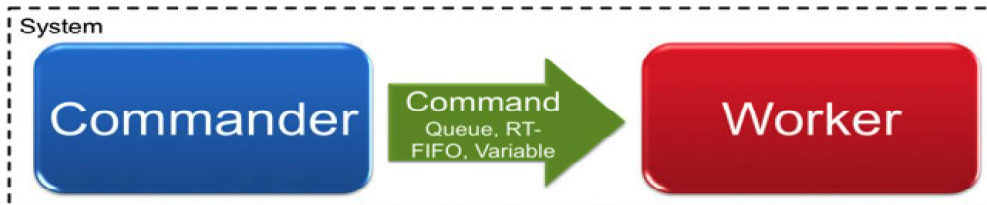


图3.16在Lab VIEW中，有好几种方法可以在并行循环中传递命令，如实时FIFO、队列以及共享变量

用于命令的共享变量

共享变量为触发等操作，在并行循环间提供了一个可扩展的传送命令的机制。共享变量必须保证实时安全并且提供一个机制在FIFO中缓存命令。

创建一个单进程共享变量，并且在RT FIFO标签中，选择enable the RT FIFO，并设置为多元素的类型。

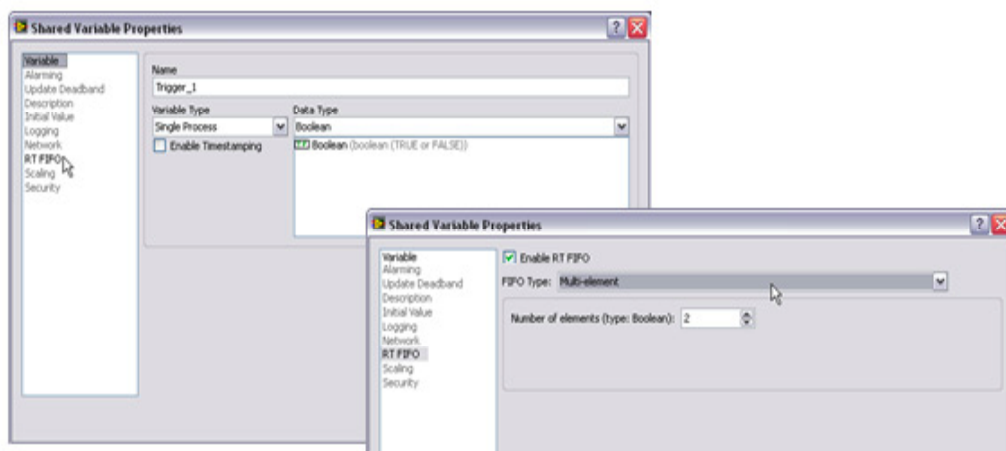


图3.17将一个可以实时FIFO的单进程共享变量设置为多元素是在循环间传递数据的有效方式

多元素FIFO的特点是它是一个缓存，每次写入在缓存里增加一个元素，每次读取则去掉一个元素。只要劳动者循环周期性的查看FIFO，将缓存设置为默认2个元素就不会错过任何触发/命令。

为查看FIFO，劳动者任务必须读取共享变量并且检查错误状态。如果FIFO是空的，共享变量返回-2220警告。若未返回警告，则FIFO非空并且返回值为一个正确的命令。

共享变量每读取一次，从FIFO中去掉一个元素（假设FIFO非空）。正因如此，不可能有多个劳动者从相同的FIFO中接受命令，但可以有多个指挥员向FIFO中提交命令。

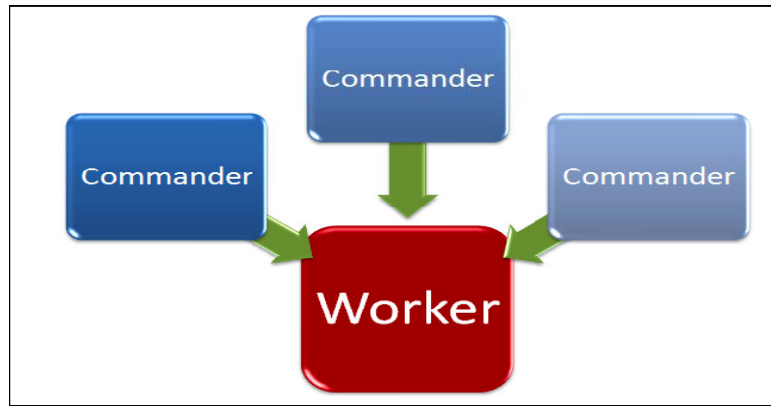


图3.18可以有多个指挥员来发布命令。但只能拥有一个工作者来执行命令

数值命令

命令比简单的触发要复杂的多。共享变量支持多种数据类型，如果它被设置为数值类型，它可以传递一系列不同的命令。Lab VIEW中的枚举型定义，是一种定义命令的很好方法，命令可以直接传递至共享变量。

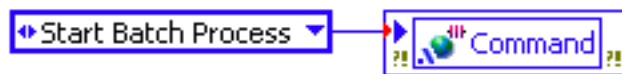


图3.19将枚举类型定义连线到数值型共享变量命令

枚举型定义提供了一个菜单式的可用命令列表。一旦定义了类型定义，任何对类型定义的修改都会自动传递到代码里。

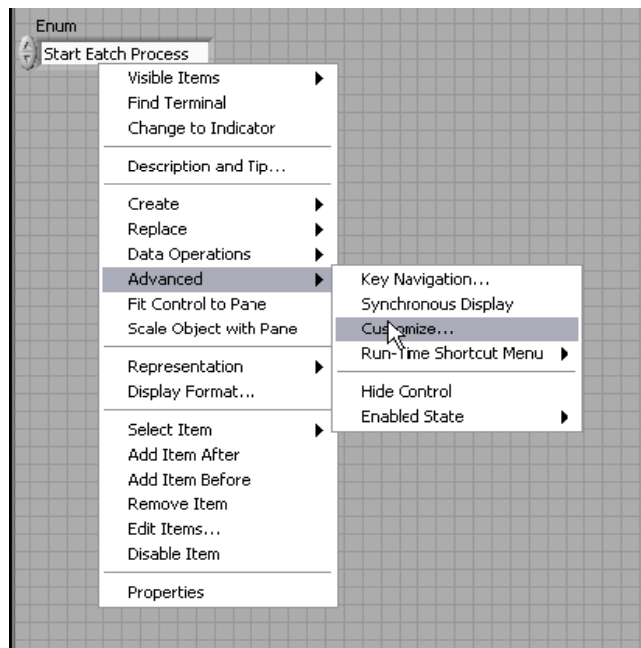


图3.20如果定义了类型定义，任何对命令列表的修改都会自动传递到代码里

使用共享变量触发一个并行循环的例子

现在修改伺服马达测试机器代码，这样当状态图表设置Output_Rotating IOV Alias为true时，触发一个并行循环。

首先，创建共享变量。在这个例子中，它是个简单的触发器，所以创建一个布尔值变量，并存储在一个新的库中，命名为Execution Control Library.

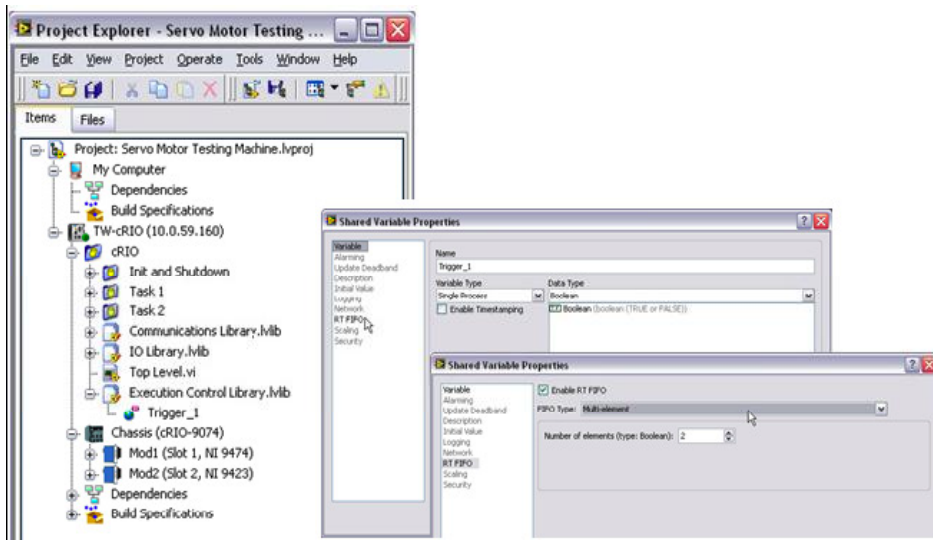


图3.21将所有的命令放置在一个新创建的命令库里，这样程序就会很有条理性

接下来修改Write Outputs Local Task 1.vi，使得当Output_Rotating输出值由False变为True时，Trigger_1写为True。

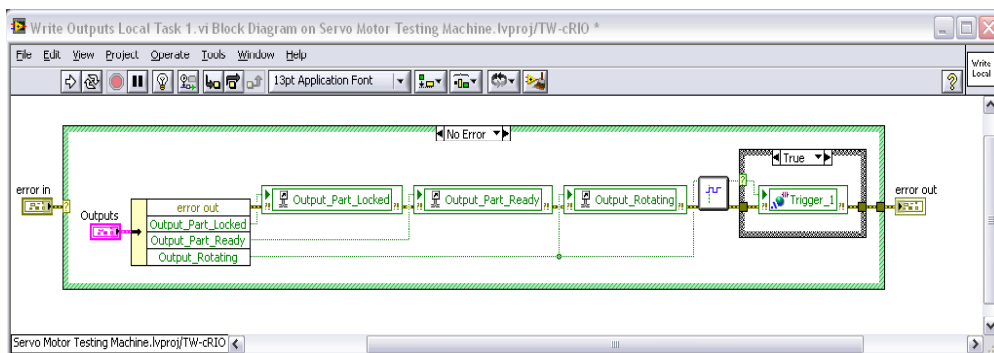


图3.22不像一般的I/O，当想要触发并行循环时，才向被当做命令使用的共享变量写入值

最后，创建一个普通优先权的并行循环并设为劳动者循环，用于读取Trigger_1和检查输出的错误簇是否含有-2220警告。由于需要重用这段代码，创建一个可重用的subVI来检查警告并存储为Empty FIFO Filter.vi。

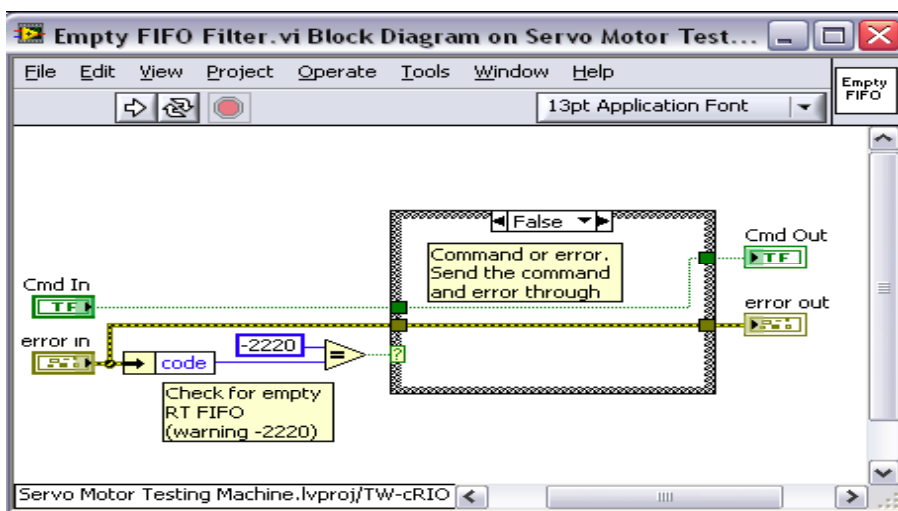


图3.23如果FIFO是空的，这个可重用的代码将起到过滤输出的作用

若并行循环从FIFO中接受到True，则触发逻辑操作。若未接收到True，并行循环等待并再次读取共享变量。

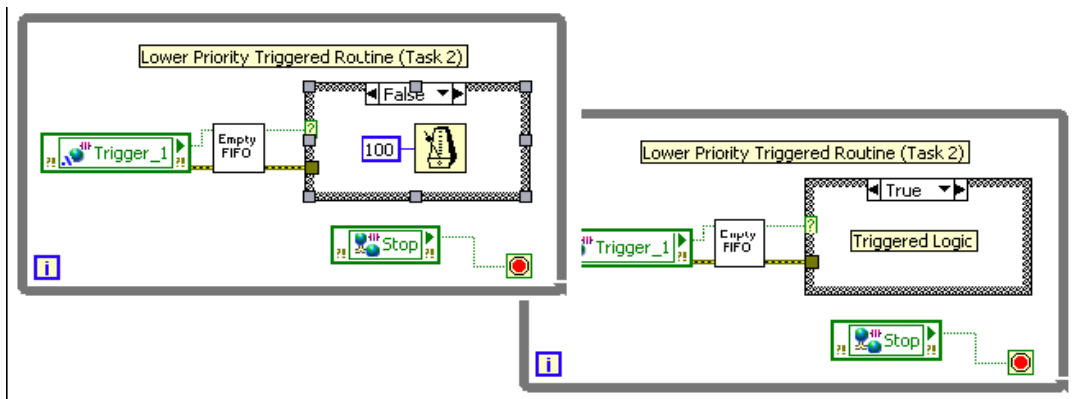


图3.24错误分支里的等待设置决定了VI的读取速率

以下是最终的程序面板，包括主程序规则和待触发的低优先权任务。

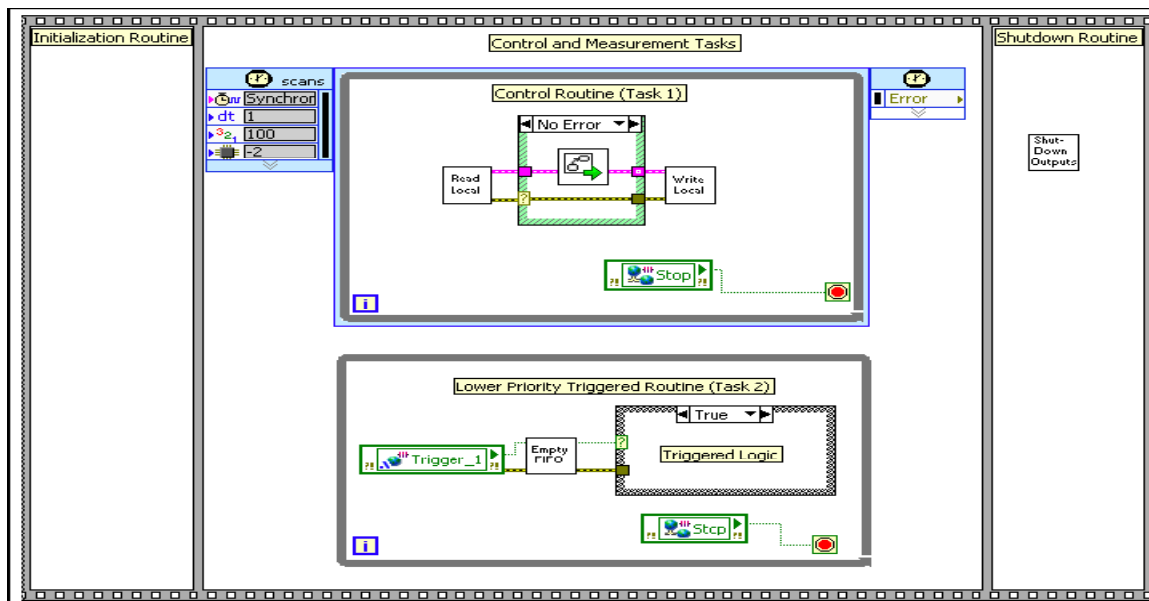


图3.2用指挥员循环（Task1）和劳动者循环（Task2）的应用

向I/O 扫描里添加数据

前一节讲解了使用I/O别名向NI 扫描引擎里读写I/O以及使用Lab VIEW共享变量在任务之间共享数据。在许多应用程序里，需要读写不被扫描引擎控制的I/O。例如，你可能需要从一个Compact RIO控制器的串行口读取一个基于RS232口的流量计的数据。另外，为一些高级I/O需求需要访问Compact RIO上的FPGA。本章探究一种架构来实现向应用程序中添加一个定制的I/O 扫描任务。

不同于扫描引擎中的I/O，需要频繁的从其他基本I/O口中访问数据，这些I/O口包括：

- FPGA
- Ethernet
- Serial
- USB

增加一个定制I/O 扫描任务(驱动循环)

如之前所讲到的，需要使用一个定时循环来向应用程序中添加任务。在Control and Measurement状态中放置一个额外的定时循环。

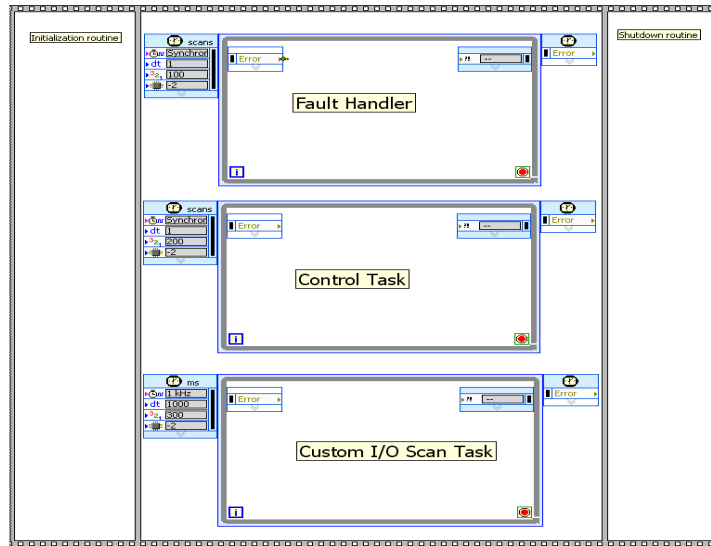


图3.26增加另一个定时循环 来来自定I/O扫描任务

优先权和定时

定制I/O 扫描任务的优先权和定时依赖于访问哪个设备和在应用程序中如何使用数据。以下是两个基本的情形。

同步确定性I/O

如果I/O驱动是确定性的且在每次重复控制任务时都要使用数据，那么应该用一个比控制任务高的优先权，来同步自定义I/O 扫描任务与扫描引擎。这将使定制I/O任务在每次控制任务重复前运行。它也允许控制任务与定制I/O中的最新数据同时运行。对于这种情况，最基本的实例应用是通过Lab VIEW FPGA模块访问I/O。

非同步非确定性I/O

第二种访问I/O设备的情形是非确定性的如并口或网络口。在这种情况下，你需要为定制I/O 扫描任务设置一个比控制任务低一些的优先权。这样可以允许控制任务确定且可靠的运行，因为它不受在定制I/O 扫描任务中I/O设备可能存在的高抖动的影响。基于必需的更新率来设置任务定时。在这种情形里，普遍的使用实例应用是从基于并行的设备或网络通信中读取数据。

接下来的例子，从读取基于RS-232流量计数据开始，学习使用非同步非确定性I/O使用实例。后面的章节着重介绍通过Lab VIEW FPGA访问I/O。

为定制I/O向内存表里添加入口

为访问定制I/O 扫描任务里读写的数据，创建能够实时FIFO的单进程共享变量，详见章节“在任务之间传递数据”。

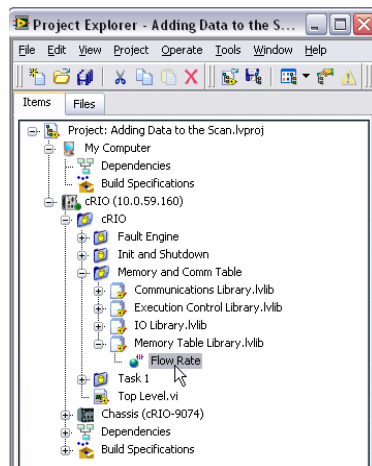


图3. 27使用实时FIFO共享变量与其他任务一起共享自定义I/O数据

增加定制I/O 扫描逻辑

增加另一个定时循环、设置了优先权和定时，并为储存数据创建了实时FIFO共享变量之后，现在可以添加逻辑来访问I/O设备。这里包括以下步骤：

1. 用I/O设备打开一个进程
2. 读写I/O
3. 关闭进程

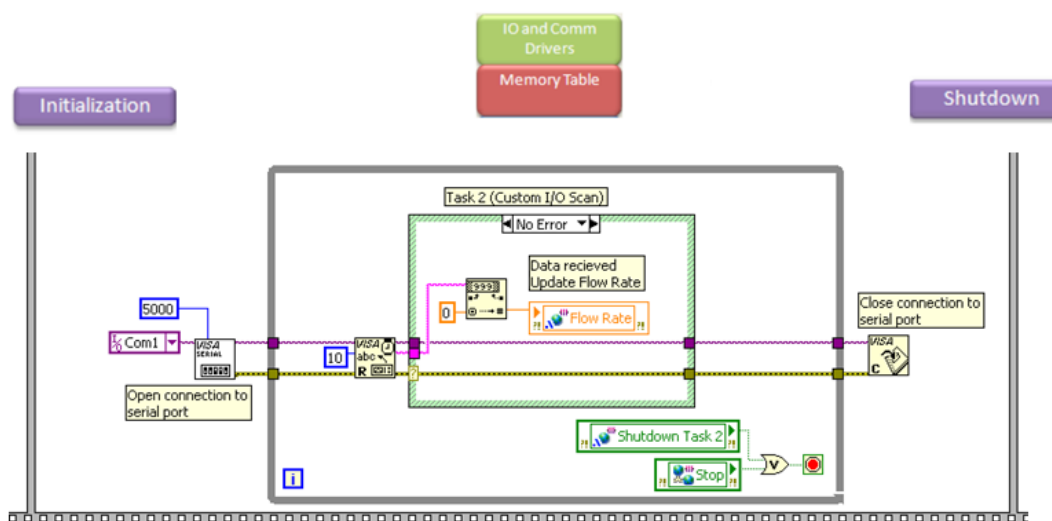


图3. 28在这个自定义I/O扫描任务里，打打开一个进程，向实时变量周期地写入数据，然后关闭进程

程序式I/O访问

在高级应用程序中，你需要运行期间程序式的访问、配置甚至发现I/O。例如，你可以通过30个I/O变量或通过一组I/O通道引用数组的索引来读取一个32通道数字模块的所有I/O值。此外，可能在应用程序运行时或当向一个已部署的系统里增加I/O模块时，需要为PWM输出配置I/O模块。Lab VIEW提供了必要的工具来为Compact RIO Scan Mode所支持的C系列I/O模块完成这些工作。这里有三个程序式I/O访问的基本使用情况。

1. 在运行时动态选择选择哪些I/O通道来读写
2. 配置I/O模块属性如电压增益及热电偶类型
3. 在已部署的系统中发现I/O模块

读写I/O

LabVIEW 2009包括有一套程序式读写I/O变量和网络共享变量的新的VI。通过在Open Variable Connection VI的共享变量句柄里创建常量并选择浏览就可以简单的提供变量的URL。

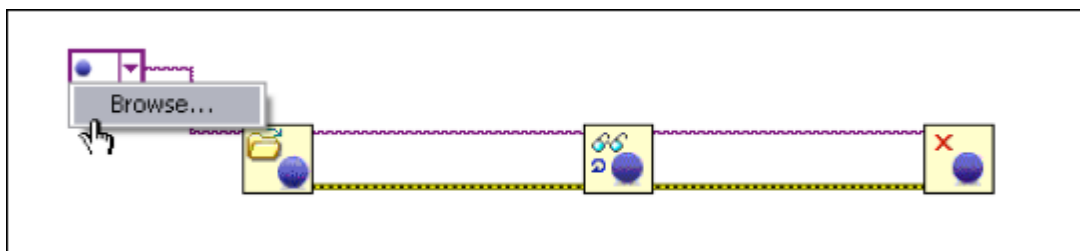


图3.29程序式地向网络变量或者I/O变量读写

可以浏览当前的变量，它们已部署在各个引擎中或者在你的Lab VIEW项目的变量里。

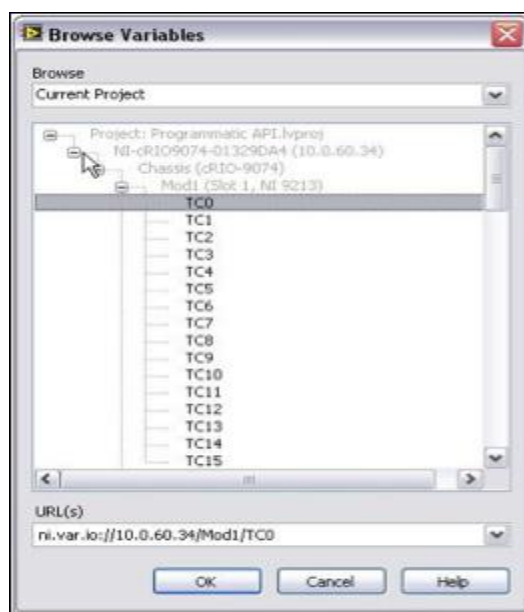


图3.30通过浏览，可以选择任何当前已配置的或Lab VIEW Project里的变量

通过创建一组通道应用的数组，就可以遍历许多通道，在每个通道上执行相同的操作，如同图3.31所示的多通道PID应用程序。

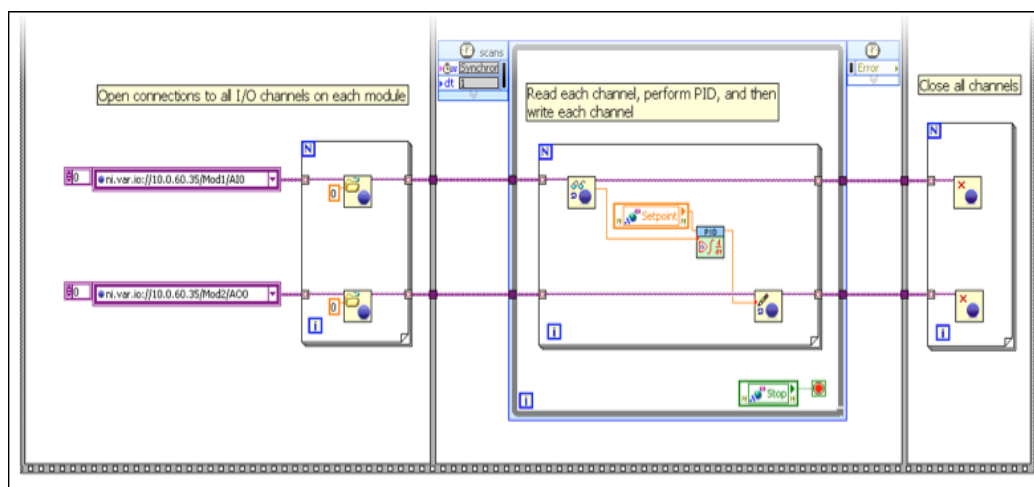


图3.31使用更加程序的编程接口像动态多通道PID应用程序来程式式读写

Open Variable Connection VI也接受用于通道输入的字符串，所以能够在你的应用程序中创建通道名并且确定使用哪些通道进行读写。

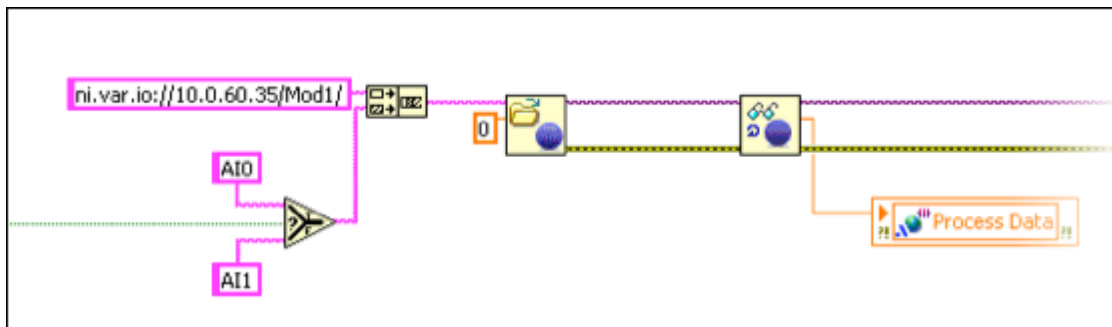


图3.32也可以使用字符串程序式地创建通道列表来读或写

配置I/O

使用程序式I/O API，在Lab VIEW项目运行时，可进入式地配置属性。例如，你可以设置电压范围，热电偶类型，PWM属性以及更多依赖于I/O模块的属性。这样就能对已经配置好的系统进行重新设置，这个操作由操作员在人机界面进行。

为了进行配置，简单地将I/O模块从Lab VIEW Project上拖拽出来然后放置在程序框图上，这样就能获得每个I/O模块的引用。

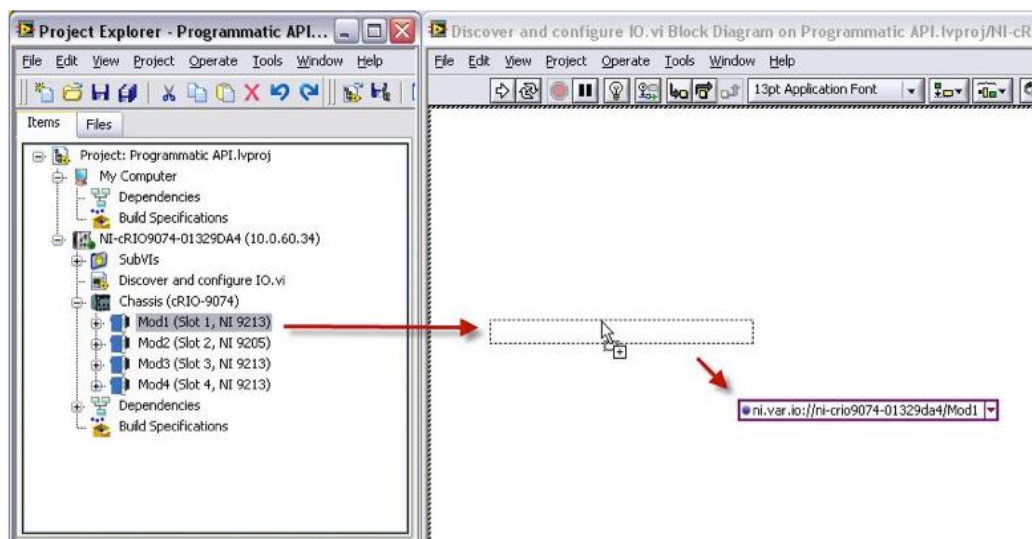


图3.33 可以使用一个模块的引用来程序式的改变配置设置

右击I/O模块URL并选择Create Property来创建I/O属性以供配置使用。在这个例子里，设置NI9213热电偶输入模块的零通道为J，这样就拥有了一个典型的J热电偶。

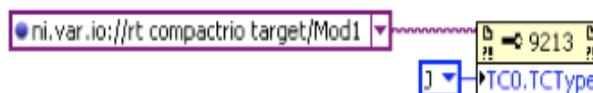


图3.34为每个模块使用属性节点来从Lab VIEW程序中设定配置

在已配置的系统中发现I/O

为了使已完成配置的系统具有最大的灵活性，可以询问系统当前哪一个I/O模块正在运行。这使已完成配置的系统能够管理和使用与最终用户系统进行热交换的I/O模块。

为了在Compact RIO系统中识别C系列I/O模块，调用NI扫描引擎函数板上的Refresh local I/O.vi（当执行控制操作时不要调用这个VI，因为它会引起系统的抖动）。一旦恢复了I/O，可以通过Lab VIEW属性节点和对象的结构层次，来读取每个I/O模块的引用。为了发

现I/O，使用class Variable》Variable Object》Variable Container并选择Children[]属性。这为系统里使用的I/O模块提供了一组引用。为了识别I/O模块，使用To More Specific Class将引用转化为RSI模块来遍历Children[]组，并选择Model和Slot属性，如图3.35所示。

图3.35 LabVIEW实时程序发现系统里当前存在的扫描模式的I/O模块

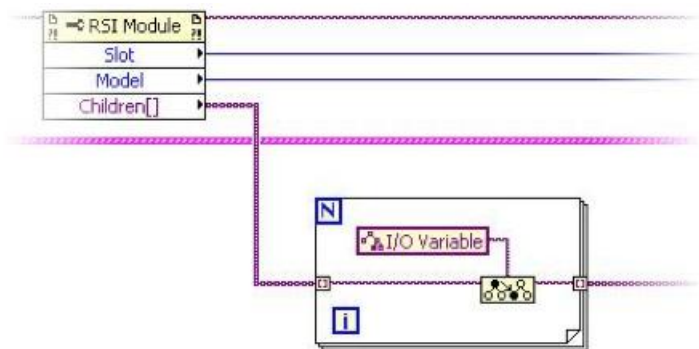


图3.36使用一个扫描方式模块的Children[]属性来读取I/O变量通道

使用这些工具，可以设计和开发一个灵活的并能自配置的应用程序。不用重新编制这个系统，程序就可以满足该领域的各种I/O的需求。

到现在为止，已经学习了一个用来获取数据并进行分析和控制的结构。对嵌入式系统来说，另一个常见的要求就是将数据保存到磁盘上，以便将来分析和检查。在这一部分里，将探究怎么创建一个简单的数据记录结构。通过这个结构，使用者可以将数据按照自定义的速率写入文件、将数据保存为一个能与其他使用者或应用程序进行转化的文件格式、并能将数据文件从Compact RIO控制器上转移到主机或者服务器上。

1. 要保存多少数据？
2. 数据保存为什么格式？
3. 被记录的数据的采样频率是多少？（即缓慢、单点测量还是波形）

这个问题的答案能够帮助选择合适的数据记录结构和存储介质包括外部USB硬件驱动、外部SD记忆卡以及通过TCP/IP到服务器的流媒体。在各种可使用的结构中，Compact RIO的平台优势就是模块化。图3.37展示了一个不同储存介质和不同数据存储结构的图标。

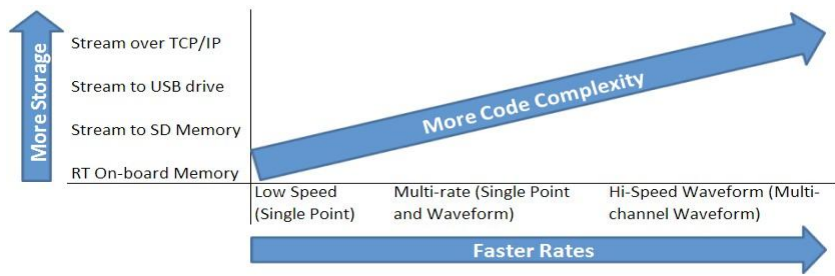


图3.37CompactRIO的各种数据记录结构

一个Compact RIO系统会自带一个高达2GB的非易失性存储器。这个存储器与计算机上的硬盘相似，但是为了提高可靠性，它没有移动的部件、可以在很宽的温度范围内使用并且使用一个叫做Reliance的更可靠的文件系统。Reliance是一个用Datalight开发的事务缓存文件系统。Datalight能够允许电源的中断，并且在电源中断之后向FTA文件系统写入数据时，它能够避免数据丢失的危险。

Compact RIO也为可移动的SD存储提供了一个传送模块。一些Compact RIO系统还拥有一个USB端口，它支持USB硬盘或者记忆棒。

本节的重点是探究与控制系统一起使用的最常见的记录结构。这节将说明怎么将低速率、单点数据写入到实时控制器的板载内存上。ni.com上还有其他结构的详细说明，但是本节将全面的学习低速率的数据记录结构，包括以下部分：

- 1. 以自定义的速率读取I/O记忆表；
- 2. 将数据写入到实时控制器的文件里
- 3. 通过FTP手工或者自动的将数据从实时控制器上转移出去。

可以使用许多文件格式将测试数据写入到硬盘上。每个格式都有自己的优缺点。这节将学习两个方式：TDMS和ASC II。

首先粗略的学习一下运行在Compact RIO控制器上并将数据保存为TDMS文件的数据记录代码的开发。

板载实时内存数据记录到TDMS文件中

技术数据管理流（TDMS）是为科学家和工程师设计的一种文件格式，它将记录好的数据组织起来保存在硬盘上。TDMS是一种二进制文件。它提供了三个逻辑层，在每个层面上都可以保存自定义的属性，这样就提供了更多的结构来监测文件。并且TDMS文件可以很容易的通过NI软件平台与其他的常用应用程序比如Microsoft Excel 和 OpenOffice进行数据交互。正因为其逻辑层次以及存取速度，所以NI公式推荐使用TDMS存储测量数据。

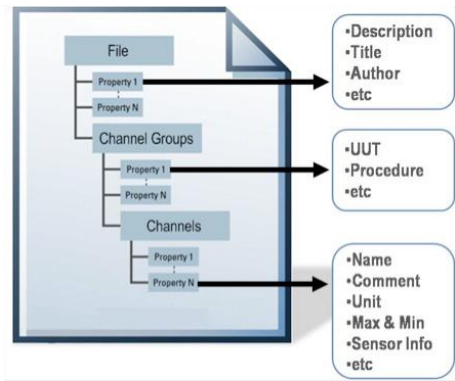


图3.38 TDMS包含的逻辑层及属性

初始化文件

首先初始化要写入数据的TDMS文件。在这一步里，选择一个文件名，这个文件名被用来决定TDMS文件存储那个测试的数据。

就像台式电脑一样，实时目标使用不同的字母来代表不同的硬盘。**Compact RIO**系统用驱动字母**C**表示内置的非易失性存储器，用驱动字母**U**表示**USB**驱动。为了在实时目标的非易失性存储器读取文件，就像台式机一样，使用以**C:**开头的路径。如果想要把实时目标上的数据记录到同一个目标的文件里，就可以使用**Lab VIEW**里的**TDMS Write.vi**并使用**C:\[foldername][filename]**作为这个vi的文件输入参数。

下面的例子中使用一个路径常量来将文件路径设置为“**c:\datalogging**”然后将其转化为字符串。在这个例子中使用“**Get Date/Time**”函数来创建日期字符串。例子中的时间字符串为“**Oct 6, 11:30:05AM.**”然后使用“**Concatenate String**”和“**Convert string to path**”函数创建文件名。最后将路径连接到“**TDMS Open**”函数。

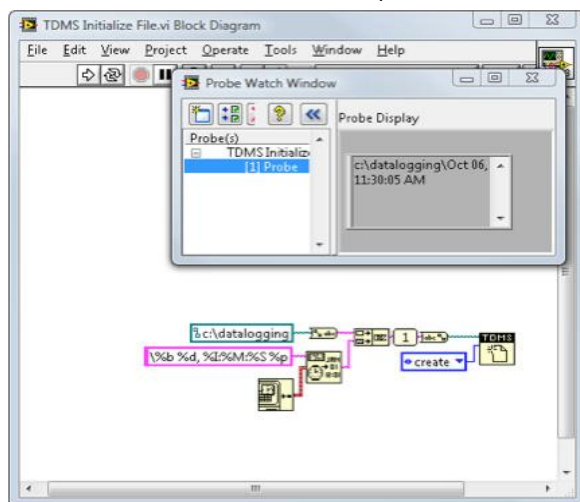


图3.39使用Proper Naming Convention for Streaming Data来初始化TDMS文件

将数据写入到文件里

程序的下一部分，即红线框所包围的区域，会将数据写入到文件的部分程序。在这节里，以自定义的速率读取保存在I/O映射列表中的当前I/O数值，并将其写入到文件。这个结构由一个与其他循环独立的While循环组成，While循环用来读取I/O数值并将他们写入到文件。结构也包含一个定时函数来确保循环以使用者定义的速度运行。

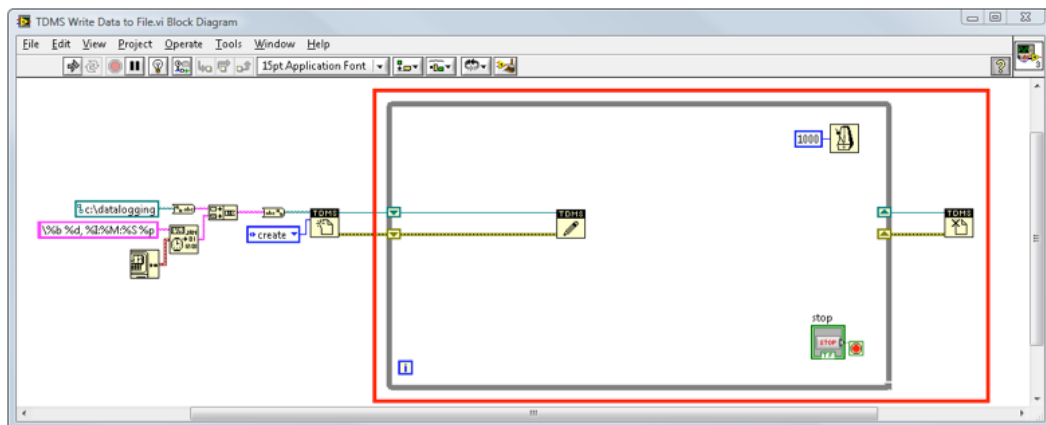


图3.40 将数据写入到文件

读取数值

第三步，即红线框所包围的区域，用来读取当前I/O存储列表中的数值并将其写入到TDMS文件里。在这一步，将要写入文件的共享变量放置在While循环内。然后将其数值写入到一个具有相同数据类型的数组里，并将其放入到文件里。

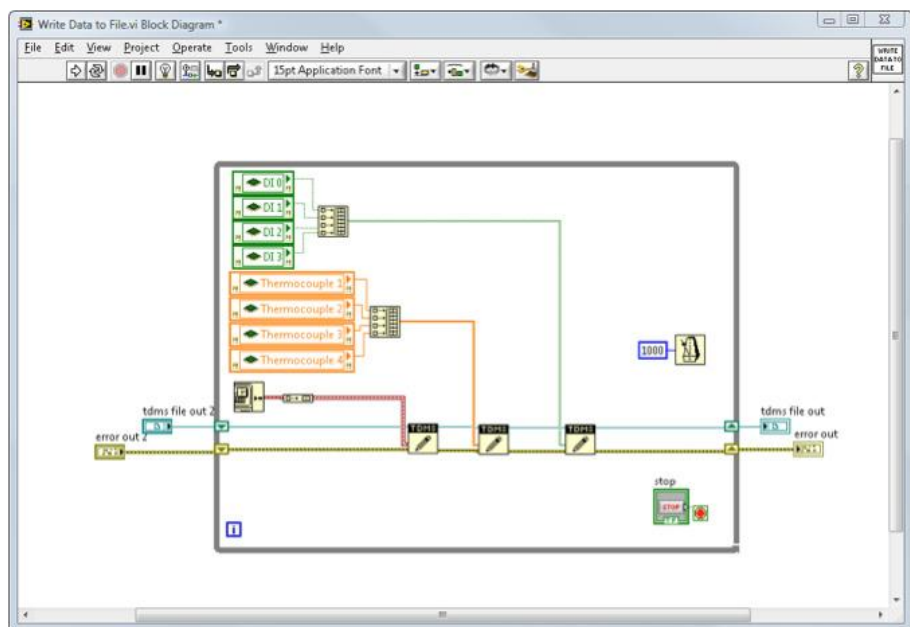


图3.41读取并格式化数据

就像前文提到的那样，TDMS文件自带三个逻辑层，每个层上都可以保存自定义的属性。为了更好的记录和组织数据，使用者可以在文件里集合数据并增加属性如传感器ID、日期等等，这个文件是基于数据类型和测量类型等创建的。访问 ni.com/tdms，可以得到更多关于TDMS格式和逻辑层的信息。

动态创建新文件

进一步修改上面的程序，来在自定义的时间间隔里，动态地关闭已经存在的文件并创建一个新文件。然后通过FTP将Compact RIO控制器里的旧数据传送到主机上进行分析、储存或者将这些数据写入到SQL服务器上。通过编写一个能够在自定义的时间间隔里动态关闭已经存在的文件并创建一个新文件的程序，就可以在程序结束之前读取先前写入的文件。为了创建这个函数，使用“Elapsed Time” Express VI来监测已经经过了多长时间。当经过了一个设定的时间段（这个例子里为1小时），程序就会关闭已经存在的文件并创建一个新文件。程序使用与“初始化文件”一节里相同的函数来实现这个操作。

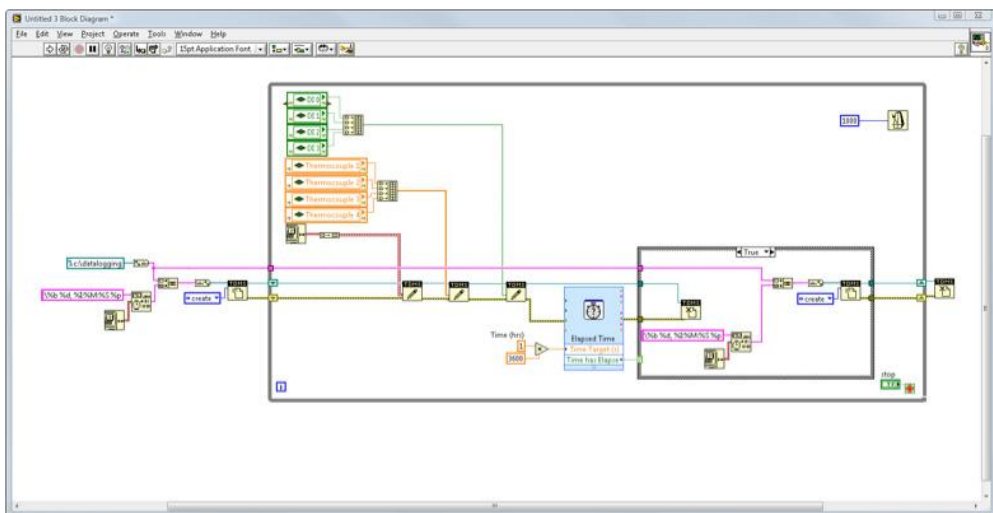


图3.42动态创建新文件

板载实时内存数据记录到ASC II文件

ASC II是一种最常见的将数据储存到硬盘的方式，它具有简单易及格式可变等优点。可以将ASC II文件读入到任何电子数据表或者Word编辑软件。ASC II文件的缺点是它比二进制文件大很多并且文件的写入速度明显很慢。

初始化文件

首先初始化要写入数据的ASC II文件。在这一步里，选择一个文件名，这个文件名被用来决定ASC II文件存储那个测试的数据。

就像台式电脑一样，实时目标使用不同的字母来代表不同的硬盘。**Compact RIO**系统用驱动字母**C**表示内置的非易失性储存器，用驱动字母**U**表示**USB**驱动。为了在实时目标的非易失性储存器读取文件，就像台式机一样，使用以**C:**开头的路径。如果希望把实时目标上的数据记录到同一个目标的文件里，就可以使用Lab VIEW里的 **Write to Text File.vi**并使用**C:\[foldername]\[filename]**作为这个vi的文件输入参数。

下面的例子中使用一个路径常量来将文件路径设置为“c:\datalogging” 然后将其转化为字符串。在这个例子中使用“**Get Date/Time**”函数来创建日期字符串。例子中的日期字符串为“Feb 23, 01:41:36 PM.” 然后使用“**Concatenate String**” 和 “**Convert string to path**” 函数创建文件名。最后将路径连接到“**Open/Create/Replace File**”函数。

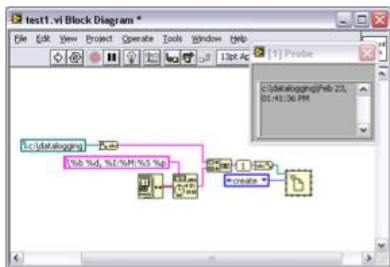


图3.43使用Proper Naming Convention for Streaming Data来初始化TDMS文件

将数据写入到文件里

程序的下一部分，即红线框所包围的区域，将数据写入到文件的部分程序。在这节里，以自定义的速率读取保存在I/O映射列表中的当前I/O数值，并将其写入到文件。这个结构由一个与其他循环独立的**While**循环组成，**While**循环用来读取I/O数值并将他们写入到文件。结构也包含一个定时函数来确保循环以使用者定义的速度运行。

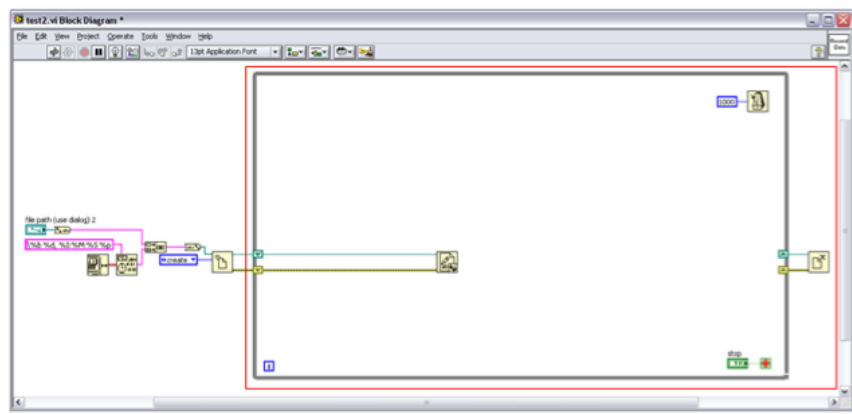


图3.44 将数据写入到文件

读取数值

第三步，即红线框所包围的区域，用来读取当前I/O存储列表中的数值并将其转化为字符串。在这一步，将要写入文件的共享变量放置在**While**循环内。然后将其数值写入到一个具有相同数据类型的数组里，将他们转化为ASC II字符串并写入到文件里。

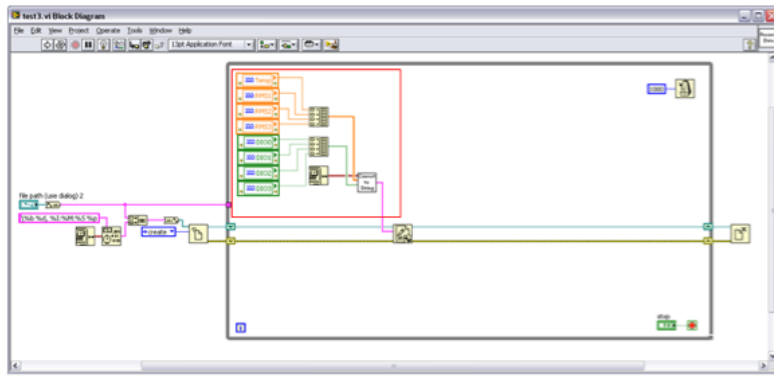


图3.45读取并格式化数据

这些变量可以是任何类型，但是在将他们写入到文件之前，必须将他们转化为字符串。这个例子中的转化在子VI里实现。最后在创建一个单字符串，这个单字符串包含每个I/O的数值，并使用“空格”或“逗号”作为分隔符，然后将其写入到ASC II文件里。将“空格”或“逗号”放置在数值之间来表示数值之间的隔离，并且在每行数值的结尾放置一个“End of Line”字符来表示行的结束。这允许电子数据表编辑器从文件中提取行和列的数据。

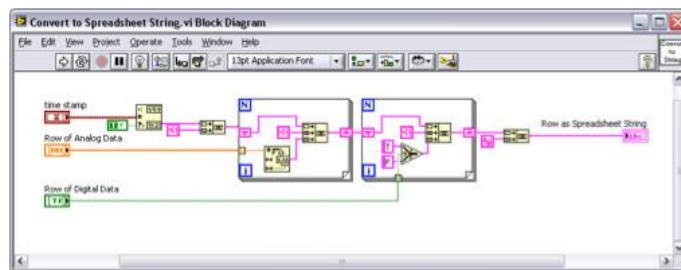


图3.46 将数据格式化为ASC II字符串

动态创建新文件

最后，在程序里增加代码来在自定义的时间间隔里，动态关闭已经存在的文件并创建一个新文件。然后通过FTP将CompactRIO控制器里的旧数据传送到主机上进行分析、储存或者将这些数据写入到SQL服务器上。若果持续向一个单一文件写入数据，在控制器结束之前不可能将数据全部从控制器里拖出，这就可能导致存储空间的限制。通过编写一个能够在自定义的时间间隔里动态关闭已经存在的文件并创建一个新文件的程序，就可以在程序结束之前读取先前写入的文件。为了创建这个函数，使用“Elapsed Time” Express VI来监测已经经过了多长时间。当经过了一个设定的时间段（这个例子里为1小时），程序就会关闭已经存在的文件并创建一个新文件。程序使用与“初始化文件”一节里相同的函数来实现这个操作。

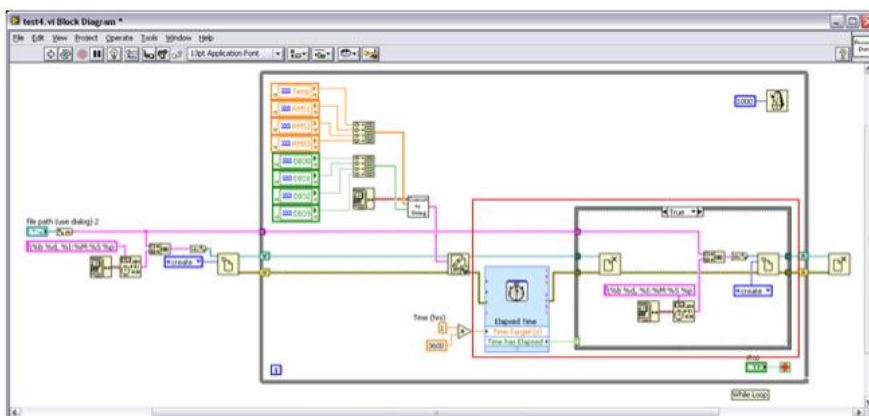


图3.47动态创建新文件

整合数据记录程序与控制结构

现在进一步深究数据记录结构，考虑怎么将数据记录结构与控制程序整合为一体。

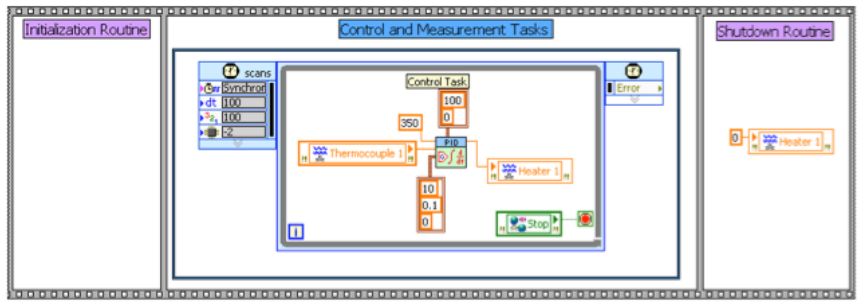


图3.48不带数据记录的控制结构

简单的将前文所展示数据记录程序增加到一个与控制任务并行的循环里，这就将数据记录增加到了控制结构里。

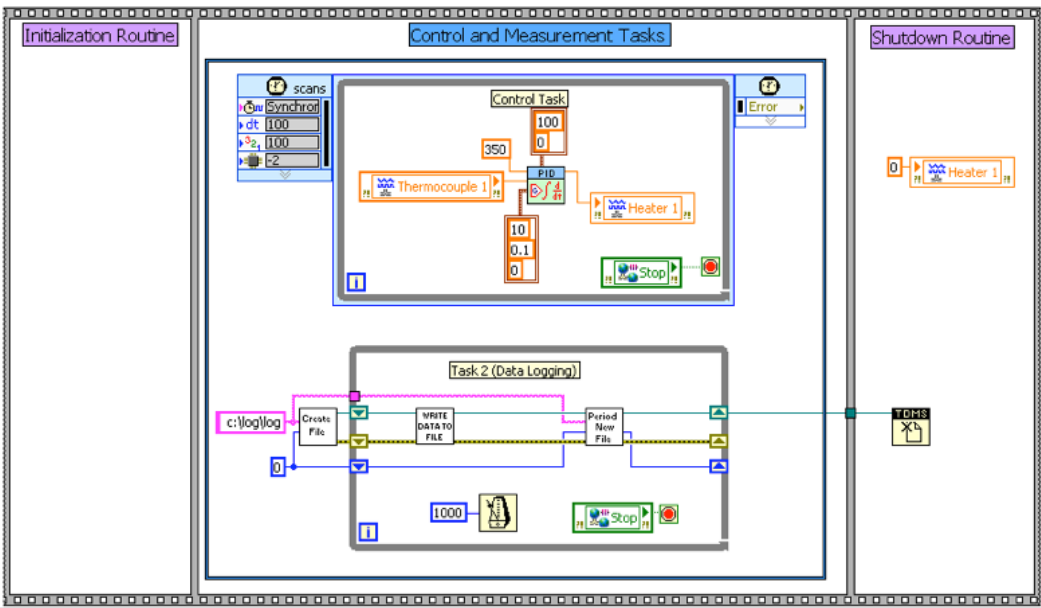


图3.49拥有数据记录的控制循环

检索记录的数据

Compact RIO控制器运行一个FTP服务器，所有可以从任何包含Web浏览器的FTP客户端的控制器上浏览文件。

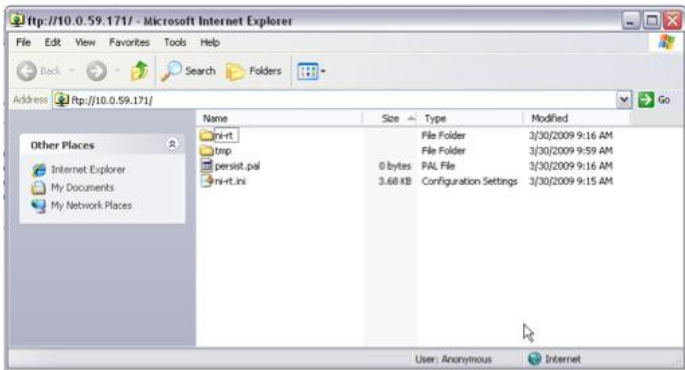


图3.50使用FTP，可以直接从Compact RIO控制器检索文件

Web浏览器

大部分的Web浏览器从访问的网页上缓存信息，所以每次访问网页的时候，Web浏览器不要下载信息，而是从硬盘上加载网页。这导致看不到Compact RIO控制器上的新文件。解决办法就是重新加载或者刷新Web浏览器上的网页，或者更改Web浏览器的缓存设置，强迫浏览器每次都检查新内容。

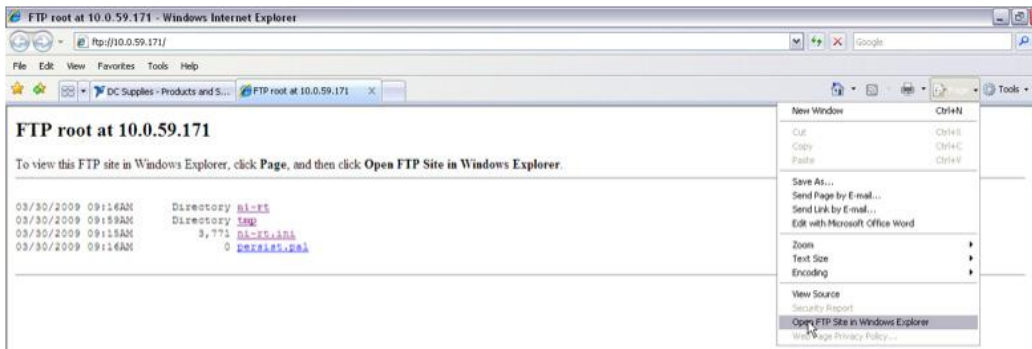


图3.51从Windows资源管理器上链接Compact RIO控制器

编程

使用Lab VIEW Internet Toolkit上的FTP VI，来自动将文件从Compact RIO控制器转移到 Windows PC上。在Windows PC运行一个程序来周期性地从控制器上拖拽新文件。下面的例子展示怎么将文件从实时目标上移动到主机上。对实时目标来说，默认的使用者是所有人，密码是一个空字符。主机是实施目标的IP地址。

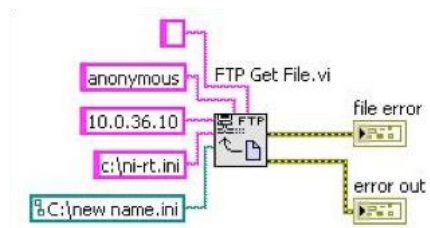
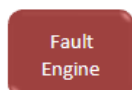


图3.52使用Lab VIEW Internet Toolkit提供的FTP VI创建Windows应用程序来编程从Compact RIO中拖拽文件

错误和故障

在**Lab VIEW**里，错误簇是一个追踪和监测错误的必不可少的工具。错误簇是一个追踪软件错误和硬件错误的完善的且良好记录的工具。对于那些需要操作员的系统来说，错误簇也是一个理想的工具。对于那些连续运行的复杂控制应用程序，可以使用一些技术来改进错误簇，使得它能在不同的并行任务以及管理循环间分享错误，并能采取相应的操作以及远程监视错误的能力。在**LabVIEW8.6**里，**Compact RIO**系统运行**NI 扫描引擎**，**NI 扫描引擎** 拥有一个叫做**NI Fault Engine**的新特征。

故障引擎



故障引擎是一个内存空间，用来在**Compact RIO**上记录和分享错误条件。通过默认设置，它可以记录和发布I/O扫描上的任何错误。也可以访问引擎里产生的新错误并编程监测任何故障。另外还可以从**NI Distributed System Manager**中监测和清除任何故障。

为使用故障引擎，需要如下步骤：

- 记录每一个错误
- 为每一个错误确定一个适当的逻辑
- 创建一个故障处理循环来监测和处理每一个错误

记录错误

使用Set Faults.vi来访问故障引擎里产生的新错误。这就需要使用一个错误簇作为输入。如果产生了错误，Set Faults.vi会自动将这个错误作为一个故障记录下来。通过创建并连线自定义的错误簇，也可以访问任何自定义的故障。在NI Distributed System Manager里，如果创建了一个自定义的错误文件并将其保存在每台机器运行的系统管理器上，那么错误发生时错误文本就会出现。

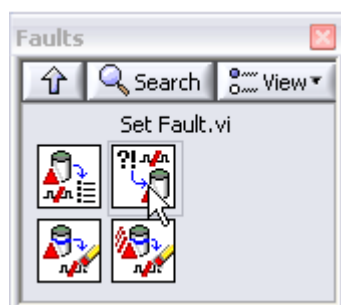


图3.53故障函数板

错误逻辑

对于每个系统来说，当错误发生时，必须决定采取一个合理的操作。错误往往表示程序的某些部位发生了错误，但会根据问题的严重程度来采取不同的操作。比如，如果错误表示热电偶发生了损坏，可能会选择通知维修，但是会通过使用其他的数值来估算温度从而继续运行控制进程。但如果错误表示机械的运动控制没有跟上命令，那么就需要立即停止机器直到维修完成。可以读取为每一个故障设置的错误代码并决定采取什么操作。在这个例子里，如果同样的错误代码出现10次，就必须停止机器的运行。

故障处理循环

也需要创建一个循环来监测任何故障并运行故障逻辑。这个循环需要一个高的优先权，这样它就可以一直运行并可以避免被低优先权的任务所架空。

使用错误处理循环的代码例子



在这一段中提供了Lab VIEW范例代码

在这个简单的例子里，从主任务里监测故障。注意如果定时循环没有按时完成，那么就创建了一个自定义的故障（表示没有按时完成逻辑）。错误管理程序读取所有的故障并运行错误逻辑。它可以通过发送命令来停止控制器的运行。在初始化程序里，需要增加程序代码来清除所有的故障并记录认识初始化故障。

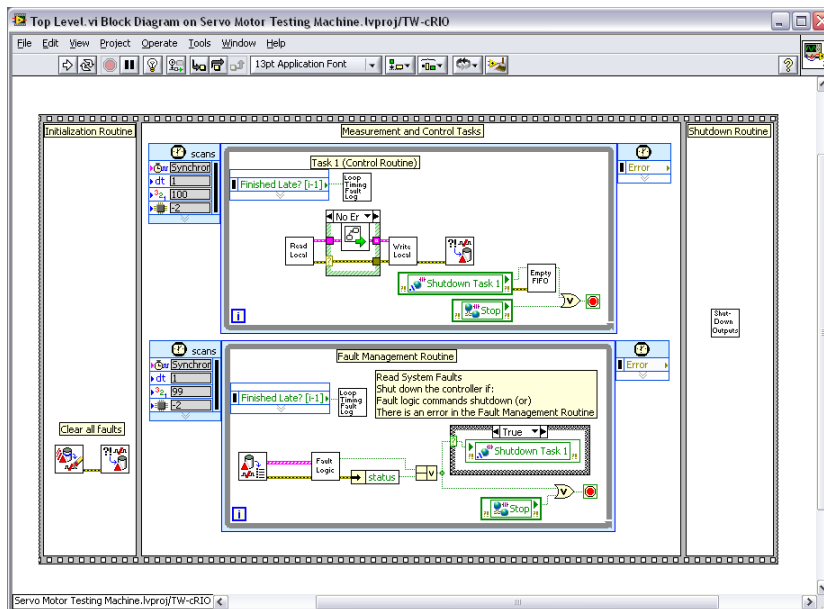


图3.54 一个拥有错误处理的完整应用程序

实时监视器

为了提供一个额外的错误安全机制来处理编程错误，比如内存溢出或者优先权倒置，所有的NI实时控制器都拥有一个内置的硬件计时器即所谓的监视计时器。如果软件没有反应，就可以使用监视计时器来运行一个恢复程序。

了解硬件-软件接口

监视计时器是一个用来连接嵌入式软件应用程序的硬件计数器，它被用来探测并恢复软件错误。在正常运行过程中，软件应用程序启用一个硬件计时器来从一个特定的时间开始并以一个设定的步长倒计时，并定义当计时器达到零时所采取的操作。应用程序启动监视计时器之后，它就会周期地设置计时器以保证计时器永远不会达到零。

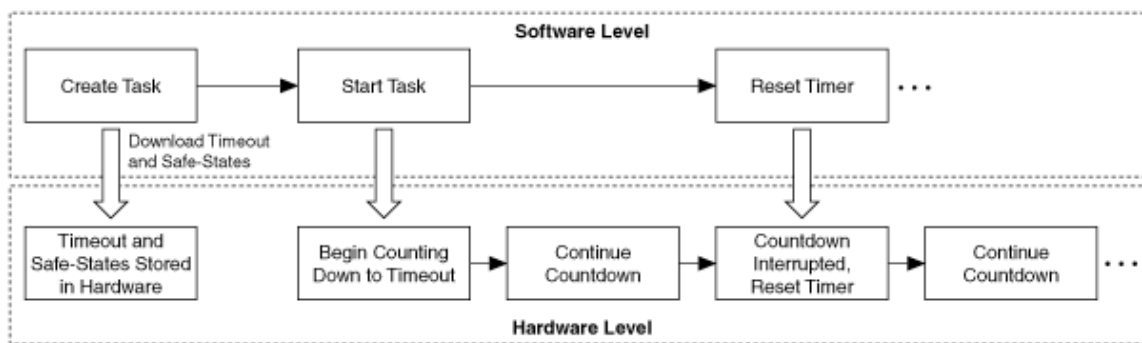


图3.55 监视计时器是一个硬件，其特征是：如果软件没有反应，它就会采取相应的操作

如果软件没能成功阻止应用程序重设计时器，那么最终就会发生超时，这是因为硬件计时器独立于软件，这样它就会持续运行直到达到零。当达到监视计时器设定的时间，硬件就会触发恢复程序。

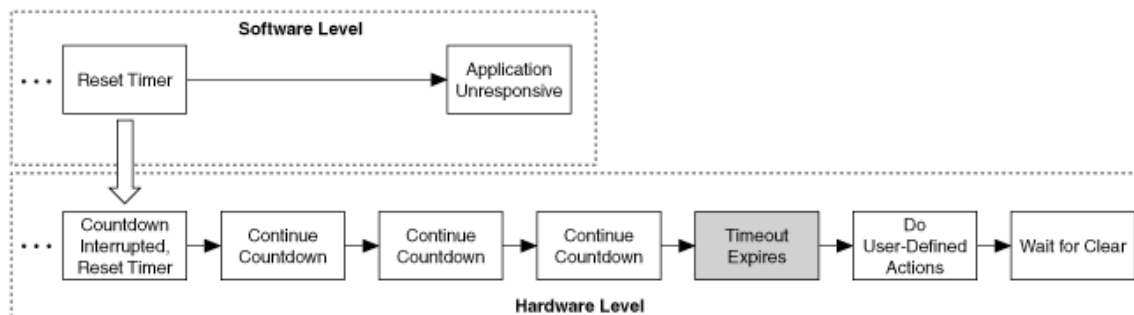


图3.56 如果软件没有重设硬件触发，那么硬件就会采取相应的操作，比如重启控制器或者中断发射

使用Real-Time Watchdog VI就可以访问内置的监视计时器硬件。配置VI为监视计时器设定一个超时值，并设定当超时发生时所要采取的操作。使用Watchdog Whack VI，在超时之前周期性地重设计数器。



图3.57Real-Time Watchdog VI函数板

选择一个恰当的超时设置

确定一个恰当的超时值范围取决于嵌入式应用程序的具体性能特点以及所需求的正常运行时间。每个系统的运行时都会发生一定的抖动，所以必须设置一个足够长的超时值，以便在可接受的抖动范围内不会发生超时。然而从另一面来说，为了满足系统正常运行时间的需求，又必须将超时时间设置的足够小，这样系统才能从错去中迅速恢复。

恰当的超时设置

确定一个恰当的超时值范围取决于嵌入式应用程序的具体性能特点以及所需求的正常运行时间。每个系统的运行时都会发生一定的抖动，所以必须设置一个足够长的超时值，以便在可接受的抖动范围内不会发生超时。然而从另一面来说，为了满足系统正常运行时间的需求，又必须将超时时间设置的足够小，这样系统才能从错去中迅速恢复。

硬件监视计时器

修改故障处理程序代码使它能够运行硬件监视计时器。在初始化程序中配置监视计时器。在这个例子中，设置计时器为1秒并设置超时操作为重启控制器。在故障循环里，“分配监视器”或者重设计数器。必须每1秒设置一下计时器，否则控制器将重启。最后，在关闭程序里，一旦关闭输出VI成功运行完成，就清除所有的计时器。

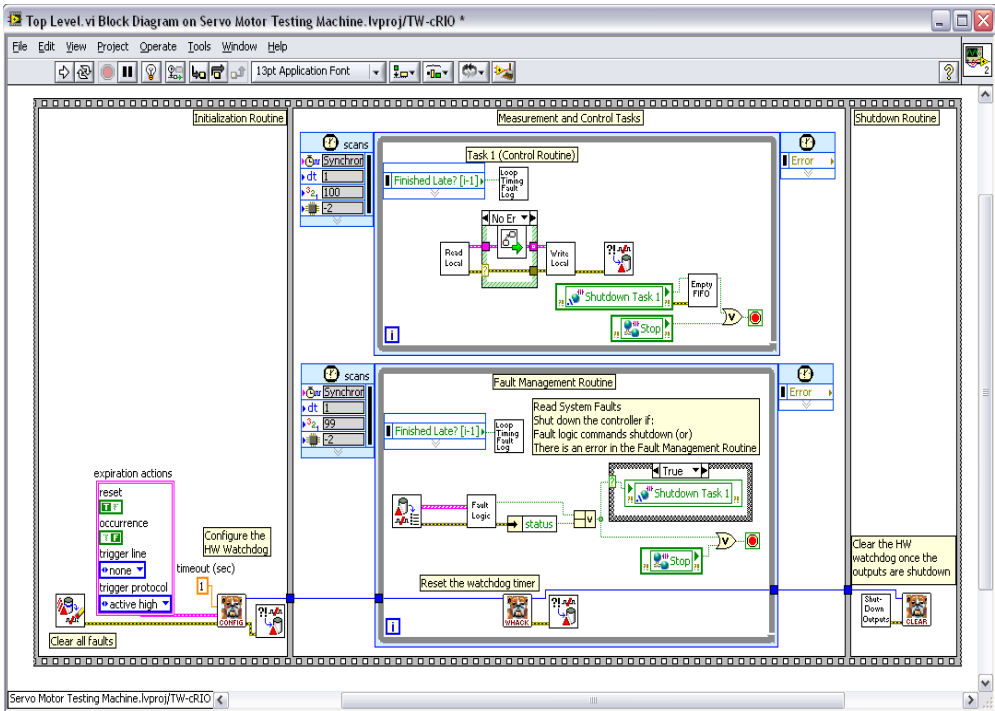


图3.57一个简单的应用程序执行一个硬件监视计时器来提高程序的可靠性

第四章

Compact RIO系统通信

通信综述

机器控制程序通常涉及到各种各样的系统，这些系统常常需要实现彼此交换信息。中央控制器可以从外部设备读取数据、接受从HMI输入的控制操作或者发送测试结果至企业数据中心数据管理库。Compact RIO硬件提供了以下几种通信选项：

- 与real-time控制器直接通信
 - 以太网（TCP、UDP、共享变量、Modbus/TCP、EtherNet/IP、EtherCAT）
 - 串行（RS232、Modbus、自定义协议）
- 利用插接式模块进行通信
 - 串行（RS232/RS422/RS485）
 - 总线
 - 现场总线
 - 原始I/O (自定义协议)

选择通信方案时，需要考虑交换信息的特性和类型。一般来说，机器控制应用程序可以包含以下几种通信类型：

- 消息型通信
- 过程数据型通信
- 数据流/缓冲型数据通信

命令或消息型通信

命令或消息型通信相对来说发生频率较小，通常被一些特殊的事件触发。例如，用户在HMI按下按钮来停止传送带，产生的消息必须由HMI发送至控制器来停止传送带传输。在消息型通信类型中，必须确保信息能够及时的传递。比如在上面的例子中，当操作人员按下停止按钮时，他希望得到及时的反馈结果（人能够感知“及时”的时间为0.1秒）。前一节介绍的利用命令型架构触发并行循环的方式，可以通过网络更改为基于消息的通信方式。

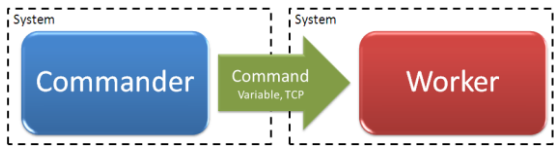


图4.1 可以使用多种技术手段通过网络包括共享数据和TCP/IP来传递数据

消息型通信方式同样适用于发送大量固定容量的数据，例如，测试结果、多变量振动监测程序以及图像帧。需要发送大量数据的大多数应用程序可以将其分解为周期性任务。在这些周期性任务中，固定容量的数据就可以通过消息模式进行发送。

过程数据型通信

过程数据通信方式通常由当前值组成，这些当前值周期性地由控制器之间传递或者由控制器发送至HMI。过程中不需要确保每次的数

据传输，因为控制器或HMI只关注最新值，而不关注以往的缓存值。可以举这样一个例子，数据传输至HMI，以显示所处于传送带的准确位置。

当控制程序需要高速采样率时，HMI程序通常只需拥有相对低速的数据更新率。因为通常是由人来观测数据，1-30Hz的采样速率是比较合适的，超过这个速率就会对带宽和资源造成浪费。对于数值显示来说，超过1-2Hz就变得难以分辨，就像给汽车加汽油时，在油泵上看到的快速变化的数字。对于图表或图形来说，人所能接受的极限更新速率为30Hz。

数据流/缓冲型通信

在缓冲型数据通信中，数据是不间断发送的，但却不是实时的。对于像波形数据这样需要捕捉各个数据点的情况，就会用到缓冲型数据通信。例如，在连续运行的MCM程序中进行振动数据传输，这些数据就被存储在网络硬盘中，用作离线分析。在这种情况下，数据容量不确定，因此很难采用消息型通信方式传输数据。这时需要打开连接，对数据通信进行连续缓冲，避免丢失数据。数据流通信方式已经超越了机器控制的基本范畴，本书将不做详细介绍。

通信类型	特点	要求
消息型	单点、 当前值	低延迟、 可靠性传输
过程数据型	单点、 当前值	最新值
数据流/缓冲型	数据连续传输	高流量、 传输可靠

表 4.1 控制器通讯类型概述

使用网络发布的共享变量进行通信

到目前为止，已经学习了使用单进程共享变量在循环间给确定性共享变量提供储存表或者在同一个控制器间发送数据。也可以选择网络发布的共享变量来通过以太网共享数据。使用网络发布的共享变量可以通过以太网在控制器、HMI和运行Lab VIEW的计算机之间实现过程数据和消息型通信

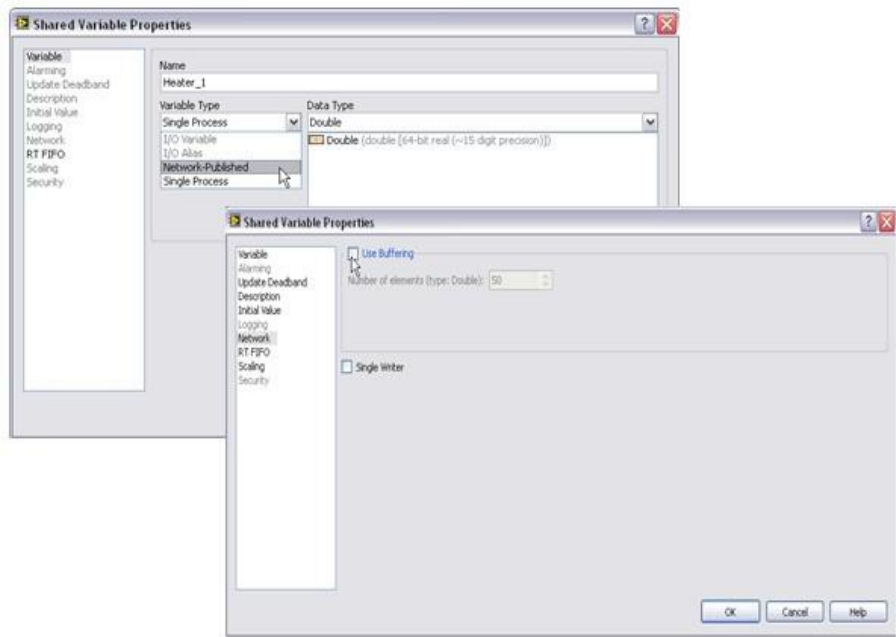


图 4.2 使用网络发布的共享变量通过以太网交换数据

网络共享变量背景

共享变量是指在网络上存在的一种软件项目，它被用来在程序、系统、远程计算机以及硬件设备之间传递数据。

共享变量的三个组成部分来使其能够在Lab VIEW上运行。三个组成部分包括：网络变量端子、共享变量引擎和发布订阅协议

网络变量端子

可以使用变量端子来执行变量的输入和输出。每个变量端子都涉及到共享变量引擎的一个网络软件项目（实际的网络变量）。图4.3展示了一个网络变量、网络路径以及工程树状图上相关的项目。

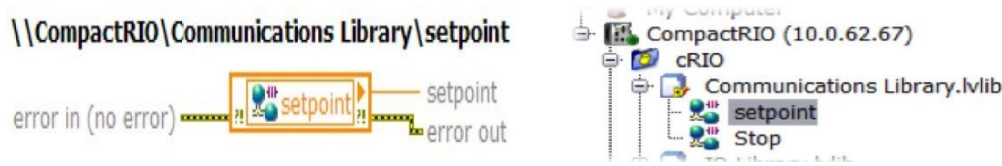


图4.3 网络变量端子及其网络地址

共享变量引擎

共享变量引擎是一种通过以太网发布数据的软件组件。引擎必须在实时目标或者计算机上或者Windows运行。在Windows系统上，共享变量引擎是系统启动上的一项功能。在实时目标上，它是系统启动时加载的一个可安装启动组件。

共享变量引擎运行时，它读入保存在非易失性储存器上的数据，并决定哪些变量需要在网上发布。

为了使用共享变量，共享变量引擎必须运行在网络上的至少一个系统上。任何一个网络上的Lab VIEW设备都可以从共享变量引擎发布的变量上读出数据或向其写入数据。图4.4展示了一个分布式系统，共享变量引擎运行在这个系统的主机界面上，多个实时控制器通过网络变量来交换数据。

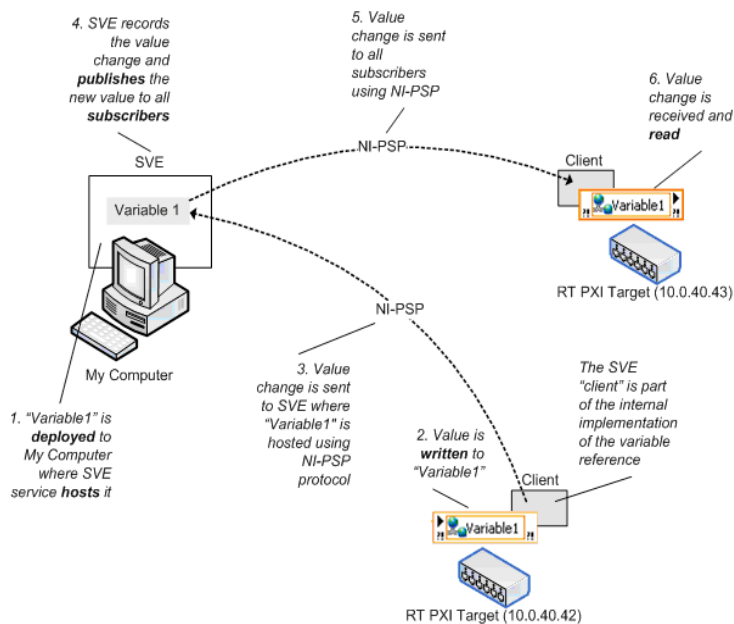


图 4.4 使用网络发布的共享变量交换数据的分布式系统

发布订阅协议

共享变量引擎使用NI公司的NI-PSP（发布订阅协议）来交换数据。NI-PSP是使用TCP建立的一个网络协议。为了通过以太网可靠的交换大量的数据，这里使用最优化的TCP。为了使每个客户订阅的个人数据所占用的以太网带宽最小化，NI-PSP执行一个事件驱动机制。在这个机制里只有数据发生了改变，这些数据才被发送到订阅客户那里。NI-PSP也将大量的信息压缩成一个包来使以太网得

使用量达到最小化。另外NI-PSP也提供了中心来检测断开的连接以及自动连接加载到网络上的设备。NI-PSP网络协议使用psp URLs来通过网络发送数据。

网络发布的共享变量特征

缓冲

可以使用缓冲来实现消息型的通信。缓冲不能用来实现过程数据型的通信。

当给网络发布的共享变量配置网络缓冲区的时候，实际上是在给两个不同的缓冲区配置大小，一个是服务器缓冲区，一个是客户缓冲区。服务器缓冲区就像图4.5所展示的标有SVE的框里的缓冲区，它会被自动创建并配置成和客户缓冲区一样的大小。客户缓冲区（图4.5右边所展示的）是用来负责维持队列先前值的一个缓冲区。每个网络发布的共享变量的使用者会得到自己的缓冲区，这样不同的使用者就不会互相影响。

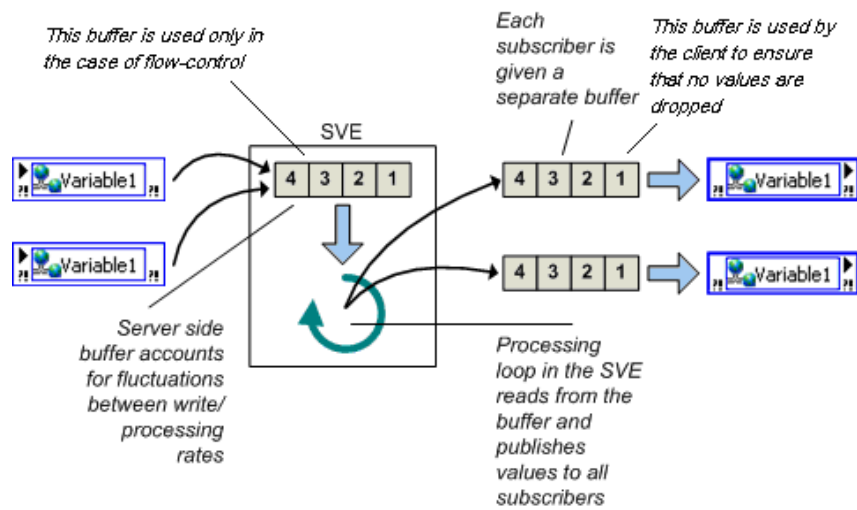


图4.5 网络共享变量缓冲区

因为缓冲给每个订阅者都分配了缓冲区，为了避免不必要的内存浪费，只有当需要的时候才会缓冲并且限制了缓冲区的大小。

决定权

网络发布的共享变量是非常灵活且容易配置的。创建一个拥有实时的先进先出的变量来结合网络发布的共享变量和单进程共享变量的功能。创建这个变量后，Lab VIEW会自动运行后面板的循环来将网络数据拷贝到实时的先进先出变量里。

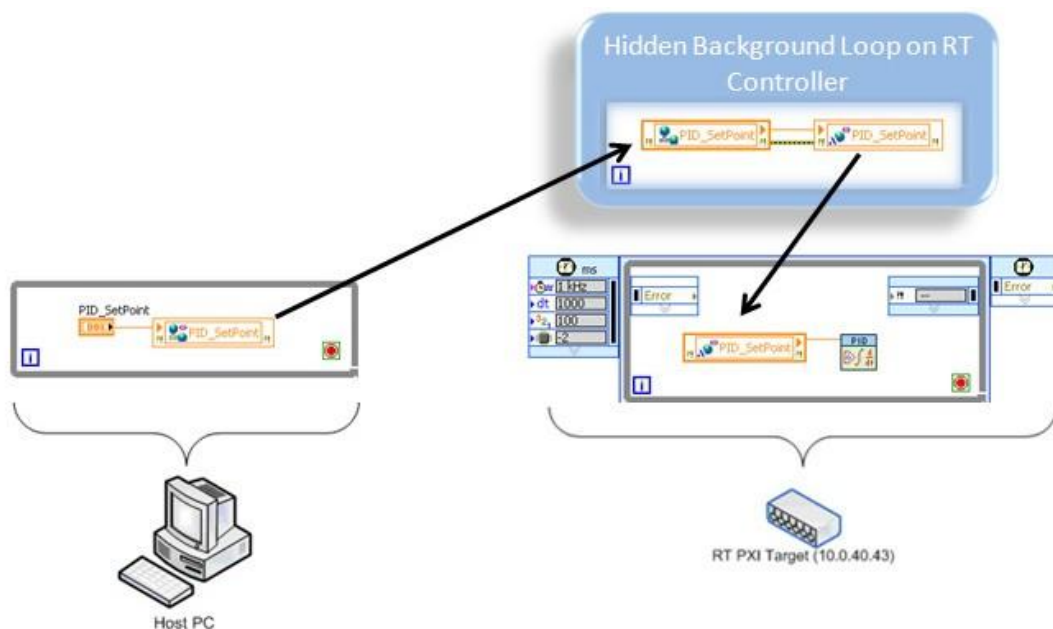


图4.6 使网络发布的共享变量能够实时先进先出后，一个隐藏的后面板循环开始运行来将网络数据拷贝到实时的先进先出变量中

虽然这个特征可以简化程序，但它也有一些限制：

- 网络发布的共享变量的一些功能不能使用
- 错误管理将更加困难，因为网络错误通过程序被传送到特殊的节点上
- 进一步修改程序来使用不同的网络通讯将更加困难

这是一个高级的特性，本文就不再检查其功能。取而代之，可以只使用网络发布的共享变量来进行网络通信并且继续使用单进程变量提供一个记忆表，这个记忆表能够储存单一控制器的不同循环间传递的确定性信息。这部分将引导你创建一个循环来将网络发布的共享变量里的数据拷贝到一个单进程的共享变量里。

使用期

所有的共享变量都是项目库里的一部分。通过默认设置，运行任何一个与变量有关的VI时，共享变量引擎就可以配置和发布整个共享变量库。停止一个VI并不从网络上移除变量。另外，如果重启拥有共享变量的机器，一旦机器完成重启，那么变量就又可以立即使用。如果需从网络上移除共享变量，必须从工程管理器窗口或者NI分布式系统管理器上卸载变量或者变量库。

监控与数据采集(SCADA)特性

Lab VIEW数据记录和管理控制模块提供了一套附加的监控与数据采集功能，这些功能被放置在网络发布的共享变量的顶部。其包含以下功能：

- 给NI数据库的历史采集
- 警报和警报采集
- 缩放
- 基于用户的安全模式
- 自定义创建的I/O服务

网络发布的I/O变量和别名

通过默认设置，I/O变量和别名被发布到网络上。通过使用与NI扫描引擎有关的普通权限，这些I/O变量和别名被以在控制器属性上设置的速率发布到网上。通过访问共享变量属性对话框可以配置I/O变量是否发布它们的状态。

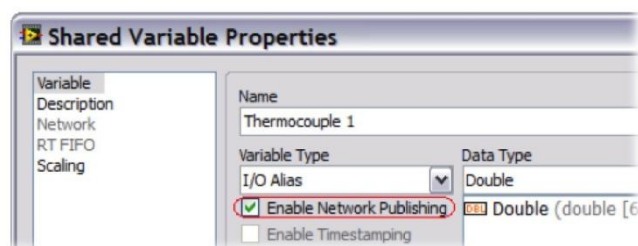


图 4.7 使I/O变量能够网络发布

为了I/O监视，对发布的I/O变量进行最优化。它不支持网络发布的共享变量的所有特征，并且不是所有的Lab VIEW设备都支持它。在不同的Lab VIEW应用程序间分享数据时，为了使操作最大限度的灵活，可以使用网络发布的共享变量。

管理和监视网络发布的共享变量

管理

为了使用网络发布的共享变量，一个共享变量引擎必须在分布式系统的至少一个节点上运行。任何一个节点都可以读或写共享变量引擎发布的共享变量。不需要安装共享变量引擎，所有的节点都可引用一个共享变量。并且对实时控制器而言，需要一个其他系统相关的可安装的小型可变客户组件。

可能多个系统同时运行一个共享变量引擎，它允许应用程序将共享变量配置在不同的位置。

当决定哪个分布式系统的计算机拥有和配置网络发布的共享变量时，必须考虑以下因素。

共享变量引擎的兼容性

分布式系统中的计算机可能不支持拥有共享变量引擎的系统，包括Macintosh, Linux, and Windows CE系统。可以在“NI-PSP Networking Technology”部分的Lab VIEW Help里找到兼容的系统和平台。

可利用的资源

系统承载大量的网络变量会消耗掉大量的资源，所以对于大型的分布式应用程序，NI推荐使用一个系统来单独运行共享变量引擎。

必须的特征

如果程序需要一个DSC功能，那么这些变量就必须被一个运行共享变量引擎的Windows主机所承载。

可用性

对分布式应用系统的功能来说，有些变量起着决定性的作用。所以这些变量必须运行在一个可靠的嵌入式操作系统来提高系统的可靠性。

动态存取的变量

网络变量也允许通过路径名引用，所以可以编程来动态选择哪些变量用来读或者写。路径名就像Windows网络共享名，比如 [\\machine\myprocess\item](#)。在这个例子里machine是机器名、IP地址或者承载变量的服务器的全部域名；myprocess包括网络变量文件夹或者变量且被称为网络变量处理器；item是网络变量名。下面是附加的网络变量引用例子：

\\localhost\my_process\my_variable
\\test_machine\my_process\my_folder\my_variable
\\192.168.1.100\my_process\my_variable

监测变量

NI分布式系统提供了一个中央位置来监测网络上的系统和管理发布的数据。不需要使用Lab VIEW开发环境，就可以从系统管理器上存取网络发布的共享变量。

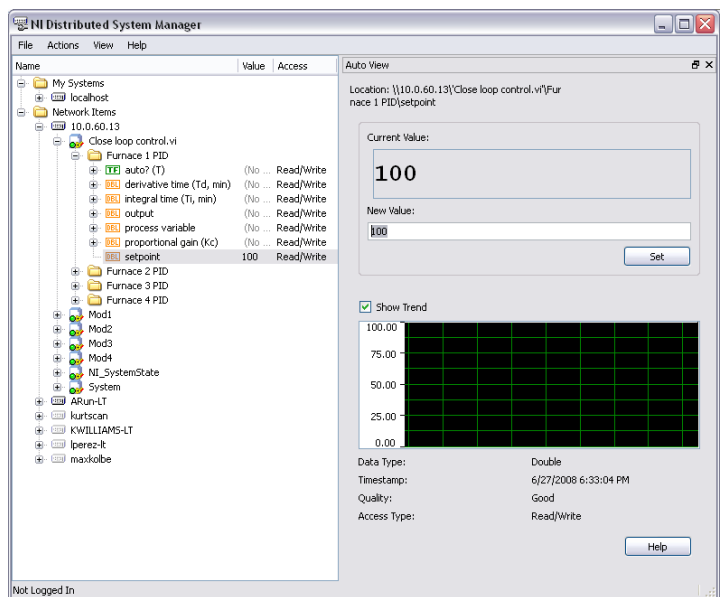


图 4.8 NI分布式系统管理器

使用NI分布式系统管理器，就可以向网络发布的共享变量写入数据，所以可以远程调整进程设置而不需要HMI。也可以使用NI分布式系统管理器来监视和管理控制器的故障和事实目标的系统资源。点击Lab VIEW的工具，选择工具菜单上的分布式系统管理器来运行系统管理器。

使用网络发布的共享变量来共享进程数据



这部分提供Lab VIEW例子代码

可以很容易的使用网络变量来发布进程变量。在这节第二部分的PID例子里，硬编了一个PID选点。现在修改这个例子，这样就可以从网络上的其他Lab VIEW设备来不断的更新选点。

第一步创建一个叫做SV_PID_Set Point的网络发布的共享变量。进入网络标签。默认的设置是能够使用缓冲，关闭缓冲，保存通讯库的变量。如果将程序代码配置在Compact RIO系统上，那么系统就会自动配置这个通讯库，并且库里的变量可以在网络上使用。

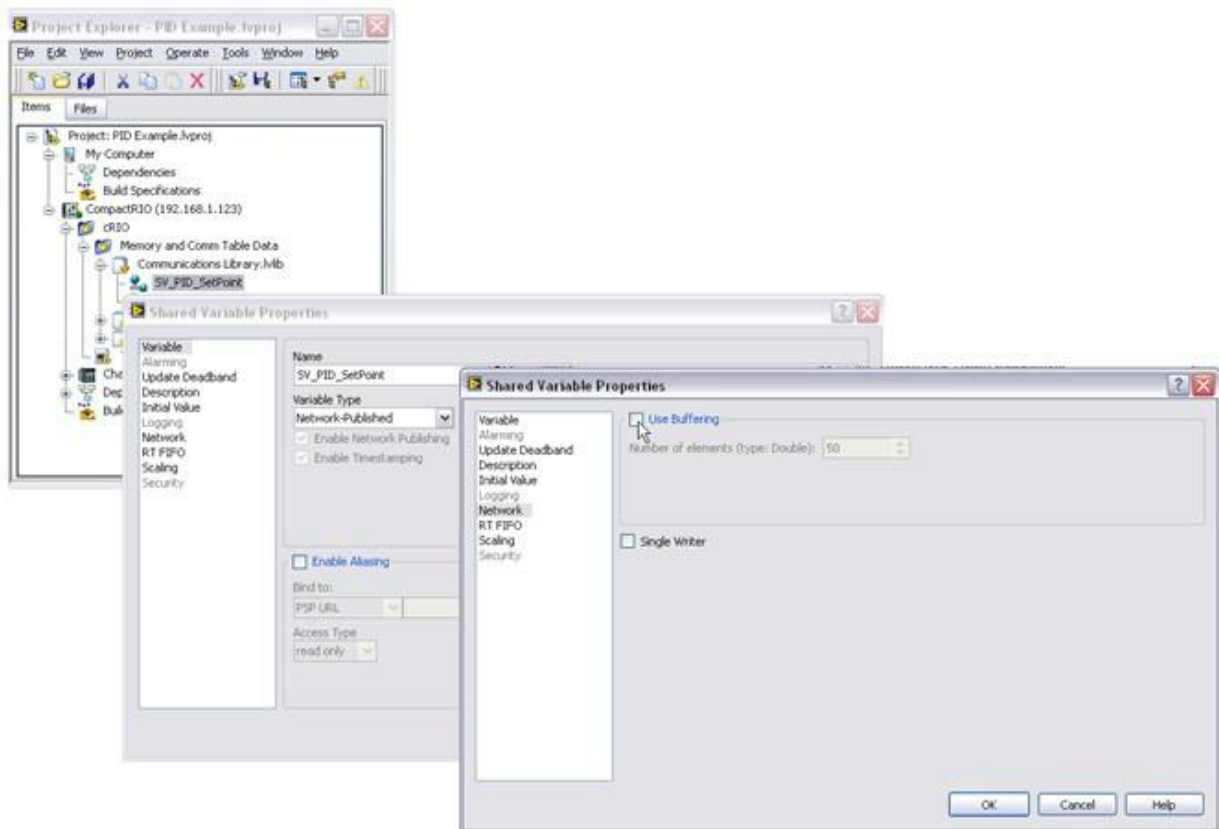


图 4.9 网络发布的共享变量允许通过以太网在Lab VIEW的节点间通讯，确保关闭了网络缓冲

因为以太网上的通讯不是确定的，所以将并行循环里的SV_PID_Set Point读入到控制循环里。同时在数据列表里增加一个元素，来将通讯循环里的数据传递到控制循环里。创建一个叫做PID_Set Point的实可以时先进先出的单进程共享变量。

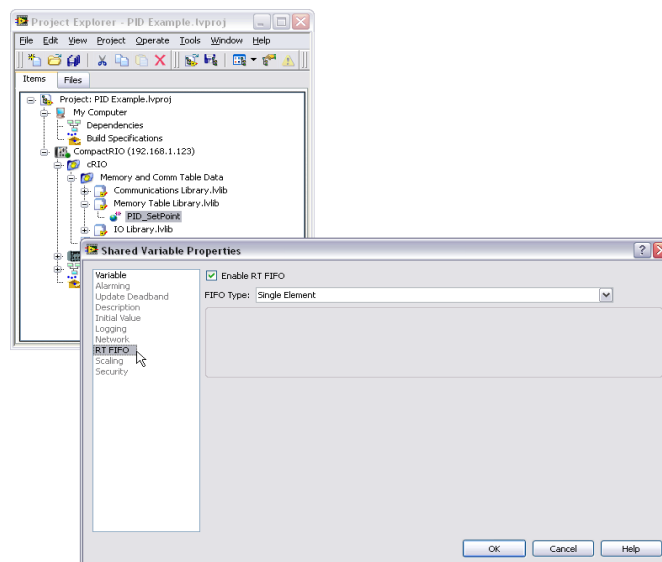


图4.10 使用一个实时的先进先出单进程的共享变量在不同的循环键稳定的通讯

现在在程序框图中增加一个普通权限的循环来读取SV_PID_Set Point，检查错误和警告，并将数据写入到PID_Set Point。以100Hz的频率运行循环。

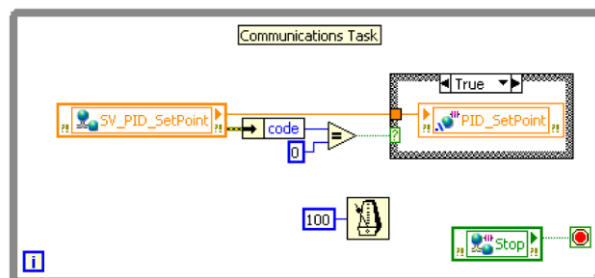


图 4.11 如果没有错误或者警告，一个循环任务将数据从网络上传递到当地的存储列表
最后设置PID框图来从SV_ PID_ Set Point上读取数据。在初始化工程中，给PID_ Set Point设置一个默认的值。

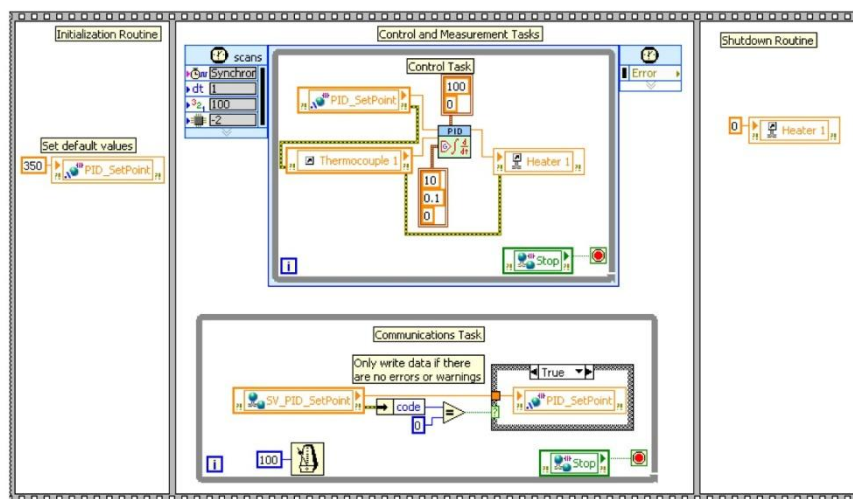


图 4.12 一个带网络通讯的最终应用程序

现在可以从网络上的任何一个Lab VIEW节点上来持续地更新设置点。

使用网络发布的共享变量来发布命令

前面的章节里讲述了怎么给一个行为比如触发任务创建一个基于命令的构架。可以使用相同的方法从网络上的其他设备来生成命令。互联网允许这些分布式系统发布他们的状态并协调他们的行为。在这种命令构架里，可以使用互联网技术在系统之间通讯。



图 4.13 使用网络变量的简单命令构架

发布-订阅模式的网络变量也使得执行多个命令变得简单。

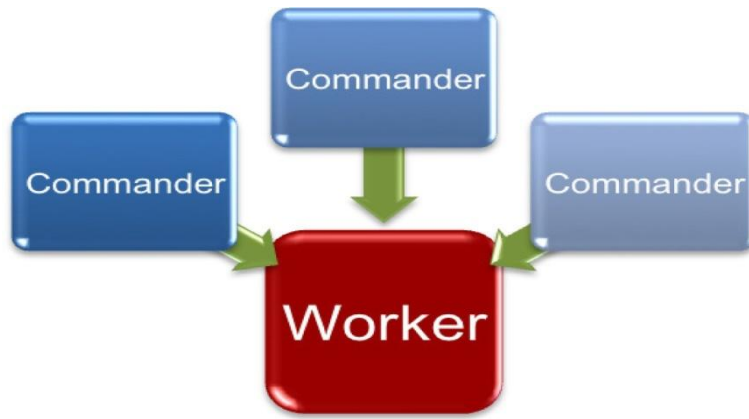


图4.14 一个工作系统多个命令

网络发布的共享变量为那些通过网络发送的命令消息创建了一个通讯通道。通过设计工作者系统，就可以创建一个可以升级的应用程序。这样工作者系统就提供了一个单独的命令解析器任务，可以使用这个解析器任务来解释和重新分配必须得命令。这样就确保新命令达到时不会打乱正在运行的关键性任务，并且这也使得修改程序来处理新增加的命令变得容易。

使用一个能够缓冲的网络发布的共享变量来发送数据。如果没有缓冲，那么连续运行时，新的命令就会覆盖其他的命令，这样就可能丢失一些命令。在使用多命令时，缓冲也是至关重要的，这是因为缓冲允许顺序的执行命令。

网络发布的共享变量在器每个执行层次都提供了缓冲机制。共享变量引擎缓冲每个变量值并将它们传递给所有的订阅者，这样每个量端子就能得到自己的缓冲。

创建一个数据类型为U32的变量，这样就可以为多个命令使用一个变量。

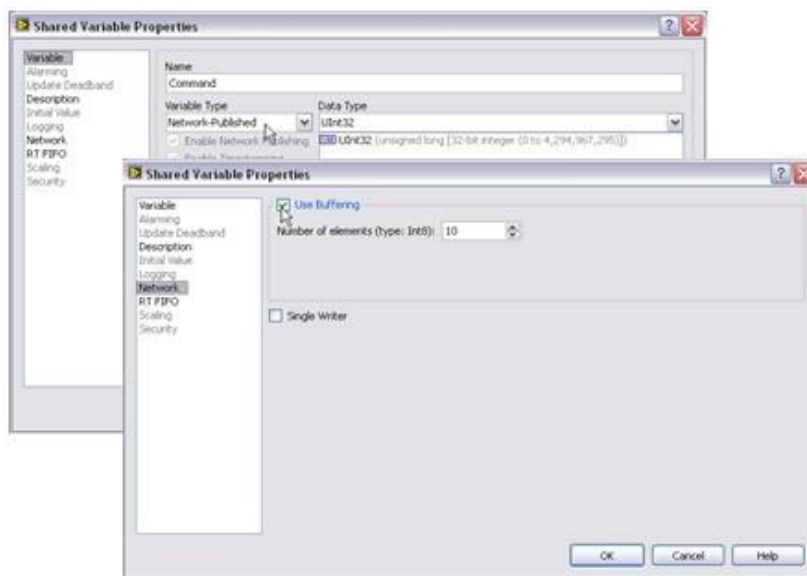


图 4.15 为网络命令创建一个能够缓冲的网络发布的共享变量

为了确保系统能够升级，就必须给系统增加新的命令，而不用大量地修改应用程序或者损坏系统的实时性能。创建一个枚举（U32）类型定义，这个类型定义为每个命令提供一个输入口，这样就更容易的维修程序。对于那些可以直接传递给网络变量的命令，用枚举类型定义来定义这些命令是一个有效方法。

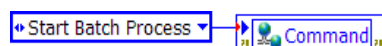


图4.16 通过网络变量来发布命令

枚举类型定义提供了一个有固定顺序的命令列表，当修改类型定义的变量时，那么所有使用该类型定义的程序都会自动修改变量。

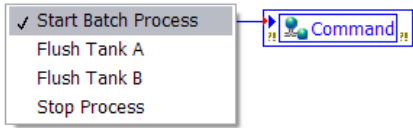


图4.17 枚举类型定义命令列表

命令构架 Commander

命令发布者是工作者所执行的任何一个命令的发布者。命令发布者就是界面事件处理器，它是HMI（人机界面）的一部分，取出界面事件并将它们转换成控制器的命令。就一个网络式的构架而言，命令发布者负责处理这些命令并作出相应的操作，比如暂时停止界面并报告当前状态。

使用标准Lab VIEW界面操作模板，就可以很容易的为那些使用者界面产生的事件执行命令构架。仅仅需要通过建立命令消息和向网络变量写入数据来将这些界面事件编译成合适的命令。

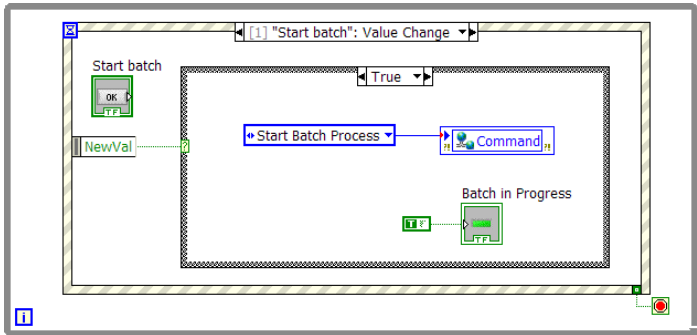
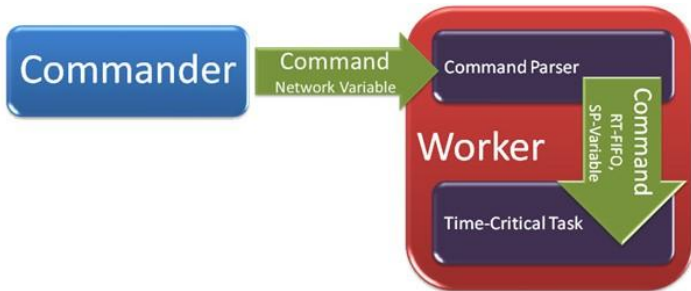


图4.18 简单的界面驱动命令发布者函数

关于建立HMI所使用的正确构架的更多信息可以在HMI章节里找到。

工作者构架

可以从工作者系统上的命令解析器任务里读取网络发布的共享变量，并检查其最新值。这种情况下，需要解析并应用输入的命令，但是不能影响控制循环的正确执行。另外，命令的执行频率不一定要和控制算法的执行频率一样。因此可以使用一个单独的任务来负责解析输入的命令并通过整个应用系统来分配必须的信息和事件。将这个叫做命令解析任务。



4.19 中心命令解析构架

简单的增加一个并行进程，就可以执行命令解析任务。这个进程接受新的命令并对其进行处理。一个单独的命令解析循环可以给系统提供可扩展性，因为：

- 它允许应用程序限制存取，来确保能够正确处理基于命令的变量、参数的范围和应用程序当前 的状态。
- 在命令被发送到任务之间。它允许对命令进行额外的处理和从新解释。
- 他允许将命令集合起来，以这样就可以将参数形成一个条理清晰的数据包，然后将其作为命令的一部分来使用。
- 可以简化程序的修改来处理不同的网络类型，比如没有配置的TCP。

- 它允许使用一个公认的命令构架，在这构架里命令和确认处理是在同一个任务里执行。

创建命令参数

超时读取

通过指定一个超时值（毫秒），来读取网络发布的共享变量，这样就可以降低CPU的使用率。从变量端子的快捷菜单上选择超时选项，使得程序能够执行超时行为。

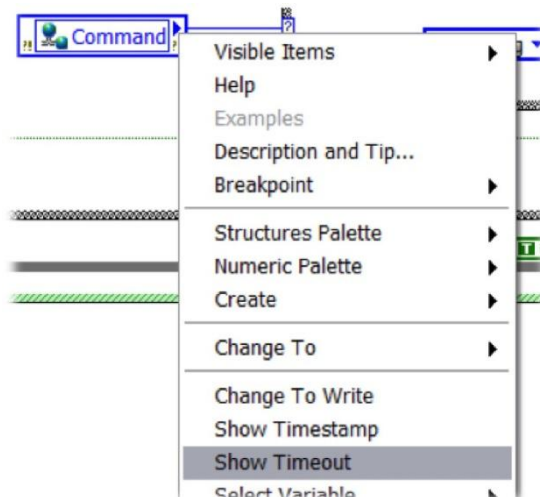


图4. 20使网络变量端子能够超时读取

在变量的超时输入端子上连接设定的超时时间，这样就可以指定在变量新值到来之前，等待多超时间。如果在超时之间缓冲里没有新值，那么变量端子就返回上一次的值并将超时布尔端子设为真。然后就可以处理超时任务。

错误处理

在这种情况下，需要考虑应用程序在错误条件下的反应。网络变量端子会返回一个警告或者错误来标识重要部分的状态。比如共享变量引擎、各自的变量承载进程和网络状态。当错误验证命令构架的接受边时，下面这些是要考虑的最重要的因素：

A 当网络变量返回错误的时候，同时也返回其默认值。给布尔变量返回假，给数值变量返回，给枚举类型定义返回其枚举列表的第一个元素。

■ 发生错误的时候让程序优先返回错误值

B 一旦建立了连接，同时就给每个变量端子创建了一个缓冲并且变量当前的值被作为缓冲里的第一个项目被传递出去。对基于命令的进程来说就产生了一个问题，因为并不能判断当前值是工作者将要处理的数据还是上一次应用程序执行时留下的数据。

■ 必须通过读取变量值不断的刷新变量缓冲直到缓冲变成空的，这时缓冲返回一个超时。

当使用网络变量执行基于命令的构架时，这些要求是构架特有的。通过使用自定义层将变量的端子封到一起，就可以执行这些要求。自定义层用来处理上面的要求，同时隐藏一些通讯通道的执行细节。可以使用下面从命令读者模板里提取的一部分程序来执行基于命令的构架。

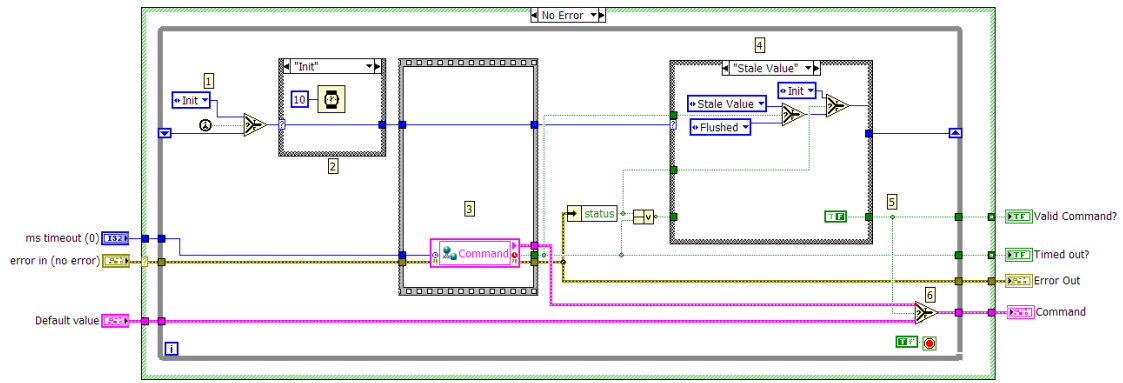


图4.21 命令读者模板函数

- 当变量端子错误时，程序执行A，返回一个特定的默认值
- 程序执行 B，通过读取变量不断的刷新变量缓冲,直到缓冲变成空的。然后等待新的命令。

使用网络发布的共享变量的基于命令的构架例子



这部分提供例子的Lab VIEW代码

现在修改一个带有通讯例子的PID来执行基于网络的命令。修改这个例子，这样就拥有一个命令解析器来停止每一个循环。

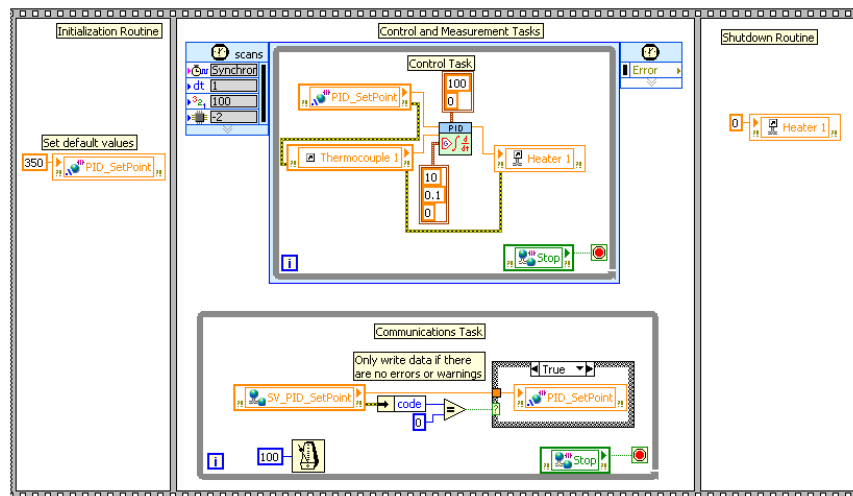


图 4.22 在例子里，修改这个应用程序来增加一个命令解析任务

首先给命令创建一个枚举类型定义。在这个简单的例子里，只需要一个简单的命令：“停止”。

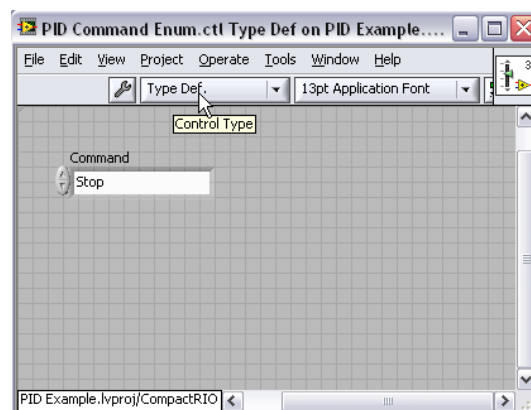


图 4.23 创建一个枚举类型定义

第二步来创建一个叫做“命令”的网络发布的共享变量。这个共享变量能够缓存。

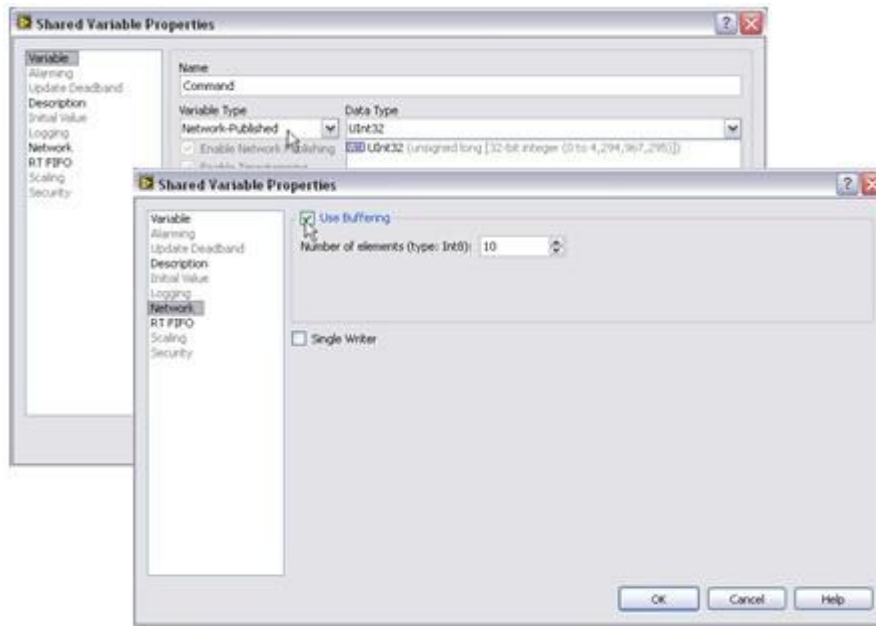


图 4.24 创建一个能够缓存的网络发布的共享变量

通过使用枚举类型定义来替换“默认值”输入和“命令”输出，来修改Command Read.vi。也需要置换共享变量的读取，使它能够通过读取“命令”共享变量。

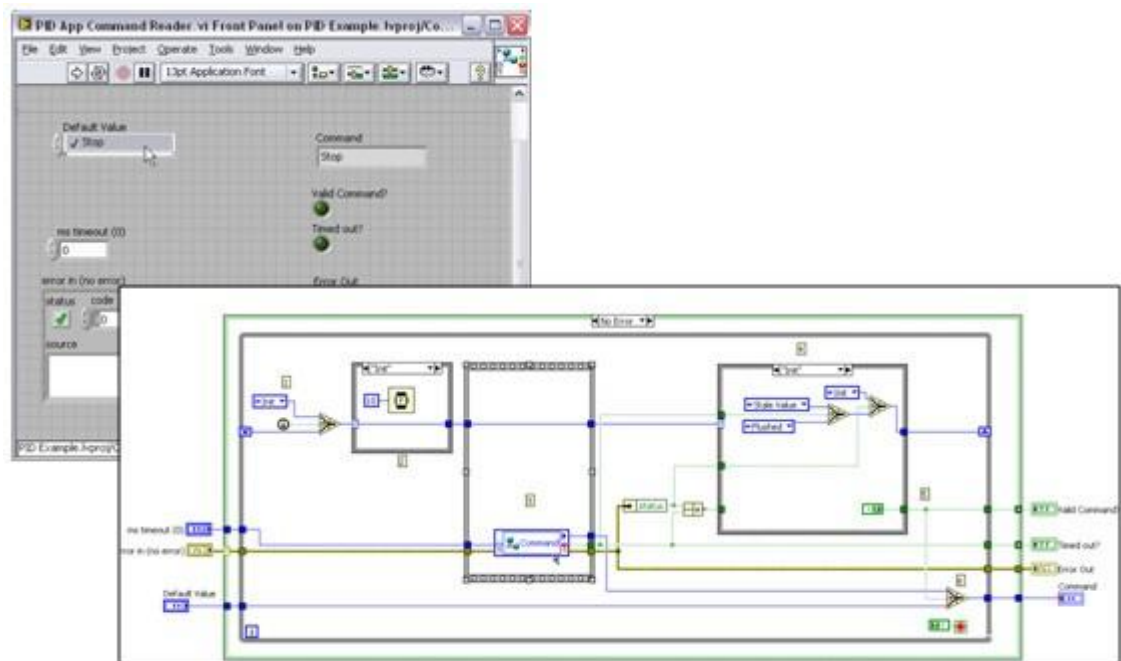


图 4.25 首次调用时，Command Read.vi清空网络队列

创建一个单进程的共享变量，将其作为停止每个循环的命令使用。

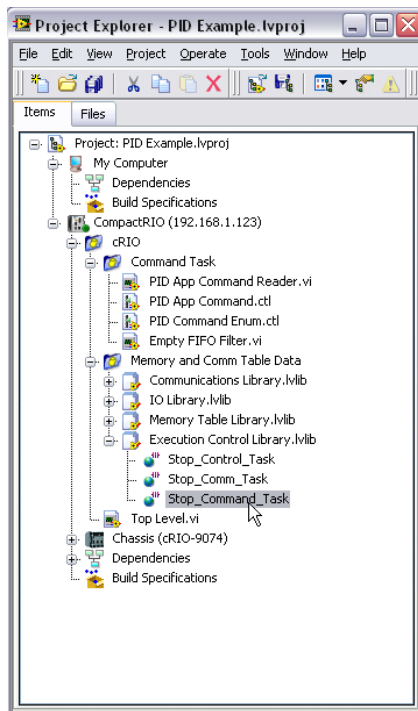


图 4.26 使用带有多个单元的单进程共享变量来在控制器内部传递命令

最后增加另一个循环来处理命令解析。将这个循环连接到选择结构上来判断命令是否合法。如果命令合法，那么为每个命令执行正确分支里的逻辑。在这个简单的例子里，只有一个命令：停止。这个命令使三个循环停止使用构架处理内部命令。

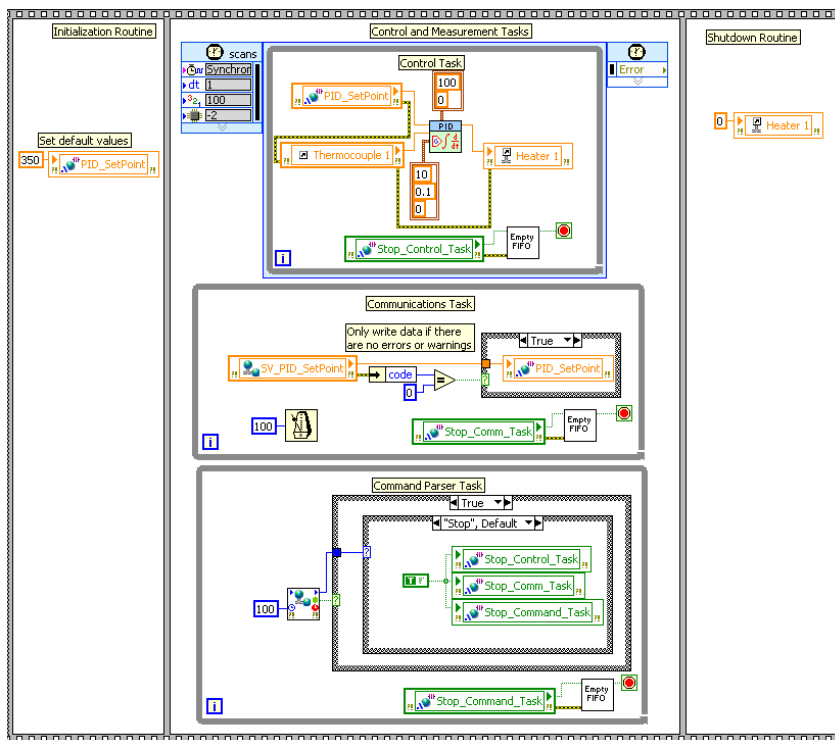


图 4.27 带有命令解析任务和停止命令的最终程序框图

基于命令的高级构架



这部分提供例子的Lab VIEW代码

基于命令的构架—包含数据和命令

扩展命令使其包含类属参数，这样增加的信息就可以和命令一起传递。

这一步能够帮助执行更多的复杂命令以及确保数据一致。

设定网络发布的共享变量的数据类型，这样就可以很容易的扩展命令消息的内容。使得命令消息包含通用数值。可以将这些通用数值作为参数解析给特定的命令。

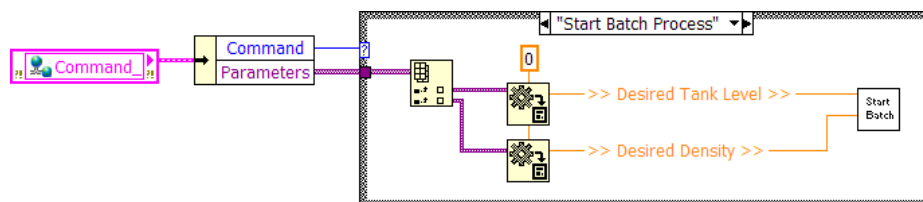


图 4.28 处理一个带有扩展参数的命令

创建一个携带参数的命令，这样就可以使参数协调一致避免类属条件。

使用一个携带命令枚举和参数数组的簇，来储存信息。可以把数组设置为数值类型，但是如果使用变体类别，就可以使用平化和非平化功能传递任何数据类型。

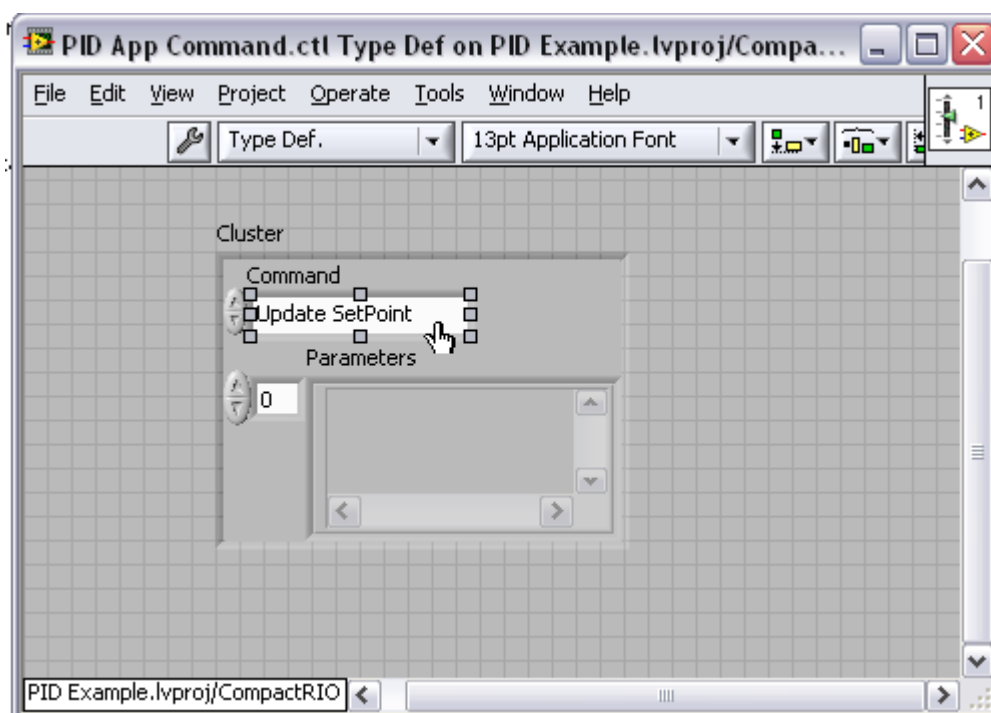


图 4.29 簇可以储存命令以及相关的命令参数

更新网络发布的共享变量的类别，使其变为簇。将数据类型转化成“From Custom Control”，并且选择创建的自定义控制，这样就可以改变数据类型。

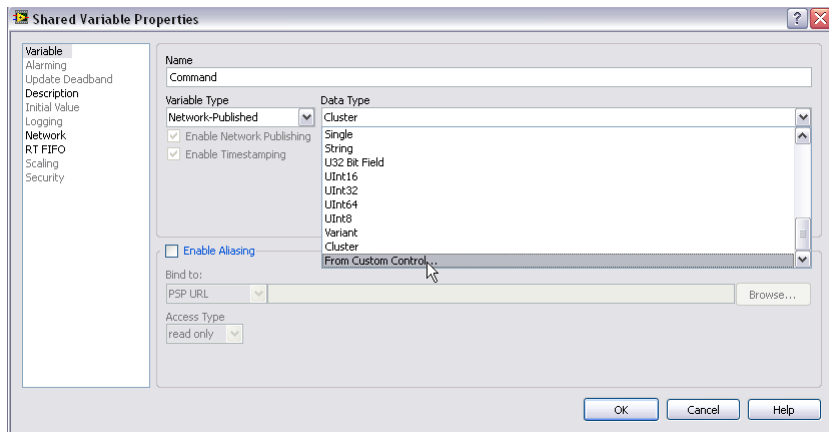


图 4.30 网络发布的共享变量可以是和命令簇匹配的自定义类型

当配置共享变量时，为了保持与自定义类型的独立性，共享变量不创建与自定义控制的永久联系。这就意味这如果改变了命令的类型定义，比如增加命令类型，就需要重新连接自定义控制与共享变量来改变共享变量的类型。可以选择为共享变量使用不同的簇，簇里的元素使用U32代替枚举，这样就可以消除这个问题。在向共享变量读出或者写入数据的应用程序里仍然使用枚举类型。

修改命令解析任务来读取命令，并将其连接到选择结构上。在每个分支里非平化参数。在这个例子里，移除了通讯循环，因为不再需要连续从网络上更新选点而是根据输入的命令更新选点。

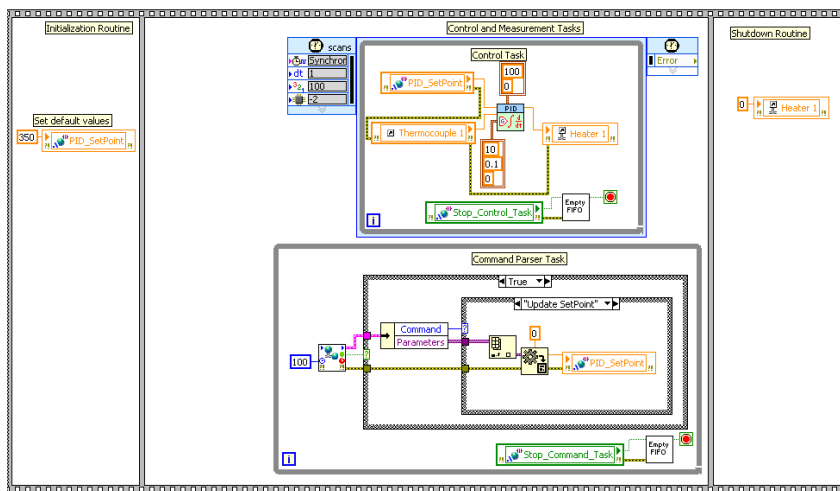


图 4.31 带有命令解析任务和相应数据的最终框图

基于命令的确定性结构

PSP协议为订阅者提供了内置的连接管理机制。变量节点返回连接信息，并且通过错误端子输出的错误和警告来返回特征状态。然而，在一些应用程序里，这也有益与执行更复杂的通讯机制来得到一个确认信息。以确认控制器 不仅得到了命令而且以执行了这个命令。为了确认信息，需要给确认通讯通道使用第二个变量。



图 4.32 使用网络变量的简单的公认命令结构

确认信息类别

不同的应用使用不同的方式定义信息确认机制。一些应用程式可能并不关心信息（不是关键信息）是否能被正确的传递，另一些应用程序可能需确认信息是否到达，还有一些应用程序需要进一步确认信息是否已经被执行了。因此，为命令结构的内容使用一下的确认信息类型：

- 正在传递：工作者循环已经从网络上读取了命令。命令被人外到达了。
- 正在接收：工作者已经评估了命令，并且考虑是否执行这个命令。
- 正在结束：一旦命令被执行了，工作者就开始工作，或者执行命令的时候以错误的形式通知命令发布者。

对于任何给定的命令，命令发布者可能选择等待任何层次的确认信息。

一个简单的执行方法就是使用枚举类型定义来列举不同的确认信息类型。

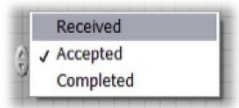


图 4.33 确认类型的枚举变量

执行确认命令结构时，除了确认类型之外，还必须考虑下面的项目。

命令ID

当确认同一个命令的多个实例时，必须给命令添加一系列数字或者相应的标识，这样就能辨认并确认任何实例的命令。

发布命令者ID

如果多个发布命令者在同时执行，那么命令就必须通过标识来指定其发布源，这样才能将确认的信息直接放回到原来的命令发布者。

考虑了以上的因素，就可以为命令信息设计一个格式，比如下面的例子。

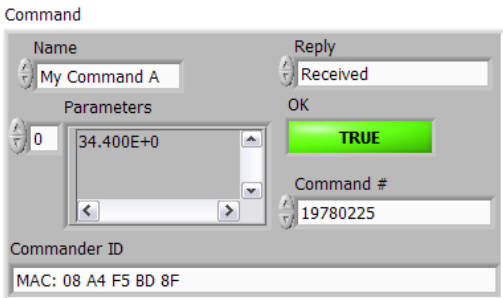


图 4.34 确认命令的采样数据簇

使用簇作为命令网络变量的数据类型，簇允许工作者接受它所需要的信息并且辨认命令以及相应的命令源。因此命令发布者需要提供与命令有关的参数以及发布命令者得ID。

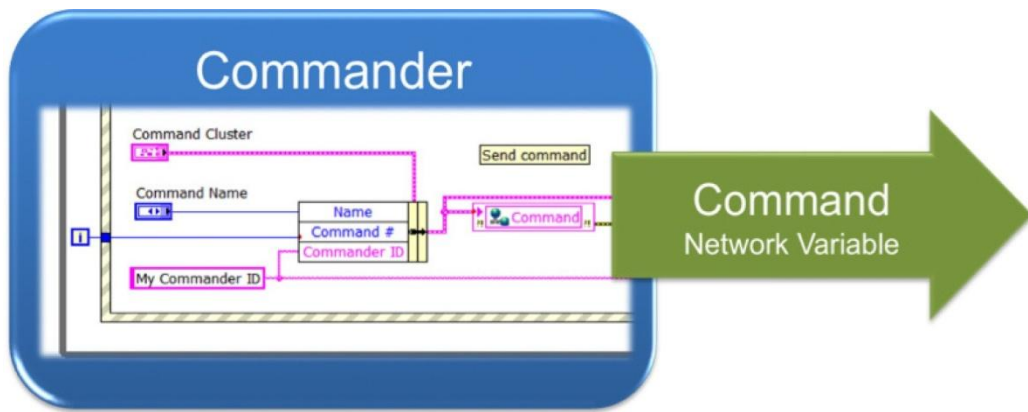


图4.35 发布确认命令

工作者使用上一节讨论的技术来接受命令并且发出确认信息。定义的簇类型定义就可以很方便的重新使用返回信息。成功接受到命令时，把信号设置成正确状态并且设置回复枚举变量的数值，这样就能执行确认信息。把信息写入到命令确认的网络变量上，这个网络变量是与命令变量一样的回复网络变量。

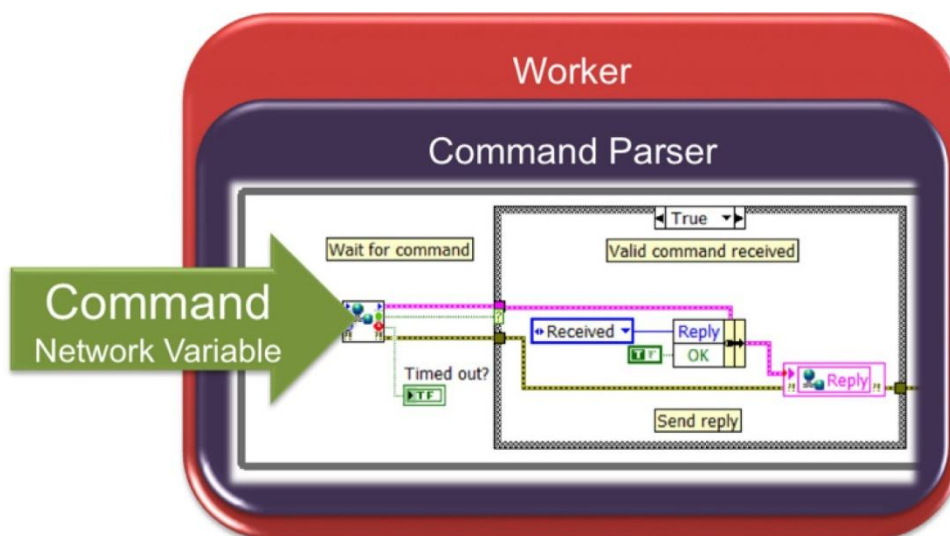


图4.36 接受和确认命令

等待确认信息是一个更加复杂的过程，因为希望得到的确认是无序到达的（不同的命令可能在不同的阶段得到确认）。这个基于命令结构的例子展示了一个VI怎么同时等待多个确认。就这个例子来说，可以创建一个简单的事件驱动命令发布者。

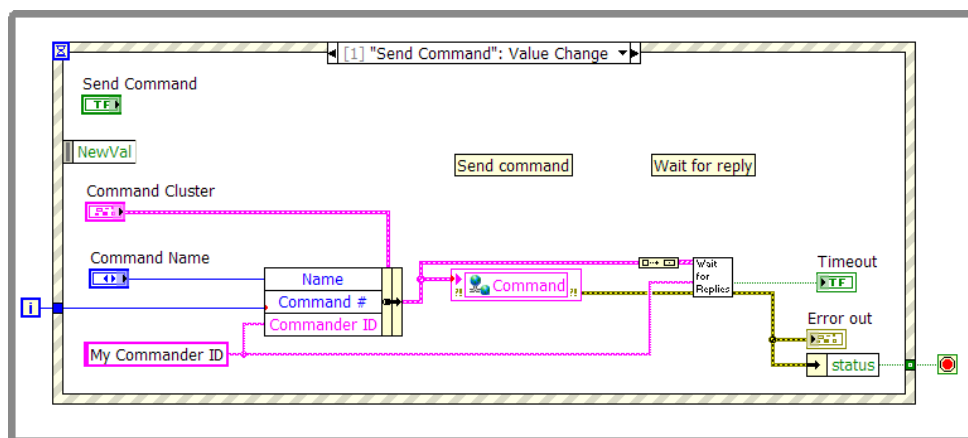


图 4.37 简单的确认命令发布VI

为命令使用网络变量需要考虑的重要事项

变量类型连接

当配置共享变量时，为了保持与自定义类型的独立性，共享变量并不创建与自定义控件的固定联系。此处自定义控件是用来定义共享变量的类型。这就意味着如果改变了命令的类型定义，比如增加了更多的命令类型，就需要重新连接共享变量与自定义控件，来更新共享变量的类型。

缓冲

因为上文所使用命令变量执行时，使用自定义控件作为他们的类型。因此有必要指出这些类型的缓冲在网络属性里所使用的单位是字节而不是元素。调整基于应用程序的缓冲的大小时，需要考虑最多有多少未执行的命令需要被缓冲储存、命令发布者的数量以及命令的平均大小包括命令里的参数。

限制命令缓冲区

缓冲的网络变量必须确保其连续的数值被保存在缓冲区里以及这些连续的数值能被发送到所有的订阅者。这就确保了每个订阅者保存在缓冲区里的数据没有溢出。缓冲区溢出经常在变量的节点层触发警告，但是在发布一订阅模型里，很难从这些警告里恢复数据。

命令历史

一个副作用就是需要刷新通讯通道来确保没有旧命令被当做新命令新执行，这需要在工作者开始工作之前刷新。一个确认的基于命令的结构通过给命令发布者提供反馈信息可以帮助缓和这个问题，反馈信息包含了命令这行者进程暂时离线时命令传递信息。

原始以太网（TCP/UDP）

当推荐使用网络发布的共享变量在Lab VIEW节点之间传递数据时，可以选择原始TCP或者UDP通讯来使用共享变量进行通讯。TCP和UDP是所有以太网标准的底层程序。所有的编程环境包括NI Lab VIEW都支持TCP和UDP工具。他们提供了底层通讯函数，这些函数更加灵活，但是不容易掌握。一些功能必须在应用程序上才能执行。比如建立连接，压缩数据。

如果需要传递通讯协议的底层控制，那么TCP或者UDP就是一个好的选择。如果需要与不支持共享变量通讯的第三方设备通讯，也可以选择TCP或者UDP，但是却没有其他的标准协议那样容易使用，比如Modbus TCP。

TCP为确保数据包传递提供了携带错误处理机制的点对点通信。UDP能够以广播的形式传递信息，这样多个设备就能接受到同一个信息。UDP广播信息能够被网络交换机过滤，但是不能提供确认性数据包传递。

TCP通讯遵循客户/服务器机制，在这个机制里服务器在一个特定的端口监测客户打开了哪个连接。一旦建立了连接，就可以使用基础写入和读出函数自由的交换数据。使用Lab VIEW里的TCP函数，所有的数据都被转化成字符串。这就意味着必须将布尔或者数值型数据转换成字符串数据来写入，读取之后还需要将这个数据再转换回去。每个信息的长度都不同，所以程序就可以判别处一个给定的信息里包含多少数值和读出多少个字节。参考Data Server.vi and Data Client.vi例子以了解Lab VIEW里的客户/服务器通讯机制。

创建自定义通讯协议

如果使用TCP或者DUP传递和接受数据，就需要创建自己的协议来定义数据打包和解压的方式。对于一个可扩展的通讯协议必须包含以下特征：

- 容易打包课解析数据

- 提取传送层（TCP/IP,UDP等）执行细节
- 当需要时再传送数据，来使网络流量最小化
- 使所有的传输量和开销这件的影响最小化
- 除了LabVIEW之外，可以把它增加到其他编程环境（C、C++等）

在每个通讯协议里，都有一些资源（元数据）被用来解析接受到的数据流。发送一套完整的元数据包会明显地增加网络消耗。因为肯能更关心应用程序的高性能，所以就想要最小化通讯对资源的消耗。通过发送一套简化的META标签信息包就可以达到这个目的。

信息和通讯处理

TCP本身就是基于信息的，所以创建的协议必须执行额外的逻辑来传递进程数据。这个逻辑就像所有周期性的发送数据一样简单，或者逻辑降低所有在网络上传递的数据。

自定义通讯协议的例子



提供例子的Lab VIEW程序代码

创建自定义通讯协议的开发人员通常也要试着为特殊的应用程序优化程序代码。这个例子没有提供创建自定义协议的详细指导。但是这个例子提供了一个写进Lab VIEW的自定义通讯协议，这个协议用来展示怎么正确的执行。如果需要与Lab VIEW之外的其他目标通讯就可以使用直接使用这个例子，或者把这个例子作为开发的源程序。

这个简单的通讯协议，简化了原始的TCP通讯的复杂度。在协议层里，当缓冲和确认完成之后，就会使用一个基于名称的机制来传传送信息。协议为每个通讯信息提供了一个报头，报头包含数据包的大小以及与携带元数据的信息相关的索引。元数据被用来预定义信息的内容。

当首次建立联系时，客户和服务端通过交换元数据来执行协议。这样就可以阻止协议传送多余的信息，使每条信息只包含6个字节。就像写入是负责将数据转化为字符串，读出负责将数据解释给信息。当读取信息的时候，可以将信息编译或者转化成基于信息的数据类型。

简单的信息协议（STM）—Lab VIEW执行程序

STM是一个通用的以太网协议。在ni.com上找到STM的安装程序，安装程序包含应用程序接口以及实例。有一些例子需要Lab VIEW实时应用程序，但大部分程序可以在任何Lab VIEW平台上运行。

元数据

元数据被当做一组簇来执行。每组元素包含数据打包以及变量解析所必须的数据属性。虽然只定义了一个名称属性，但是根据应用程序的需求来增加元数据属性（比如数据类型）就可以使用簇来定制STM。元数据簇是一个类型定义，所以增加属性不会破坏程序。

图4.38展示的例子中，元数据簇被用来配置两个变量：循环和随机数据。

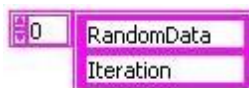


图 4.38 元数据字符串数组

在传递变量之前，会创建一个数据包，这个数据包包含了数据大小元数据ID的域以及数据本身。图4.9 展示了一个数据包格式

Data Size (32 bits)	Metadata ID (16 bits)	Data
------------------------	--------------------------	------

图4.39数据包格式

元数据ID域是指与变量相关的元数据组的索引。接收端使用元数据ID来索引元数据组以得到信息数据的属性。

简单TCP/IP信息发送应用程序接口

STM应用程序接口非常简单。对基础操作而言，它由Read VI和Write VI两部分组成。同时还有两个附加的VI来帮助传送元数据，但并不是必须使用这两个VI。每个主VI都是多态的，它允许与不同的传送层一起使用。每层的应用程序接口都非常相似。接下来的例子会简单的描述一下主应用程序接口VI。这个例子也安装了附加功能VI，可以在帮助菜单里找到他们的帮助文件。

STM Write Message vi

使用这个VI可以将任何类型的数据发送到远程计算机上。这个VI创建了一个于数据、数据名称以及元数据信息有关的数据包。当调用这个VI时，它会通过名称在元数据数组中找到指定变量的索引。然后将信息打包并通过TCP/IP使用连接ID将其发送到远程计算机上。

连接索引包含传送层索引和元数据数组。STM Read Metadata and STM Write Metadata VI.都有输出端子。

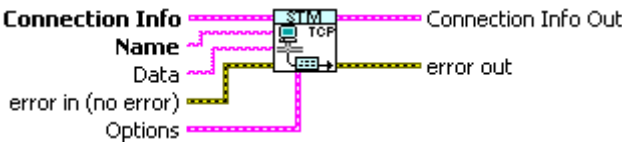


图4.40 STM Write Message vi

STM Read Message vi

使用这个VI接收从远程计算机上传来的任何类型的数据。这个VI读取和解压元数据索引和平化的字符串数据。它寻找出元数据并将它与字符串一起返回。然后应用程序就会使用名称或者其他元数据属性作为引导来将平化的数据转化成信息数据的类型。

通常在一个循环内使用这个VI。因为并不确定数据会在给定的时间内到达，所以使用“超时”来使循环周期性的运行。

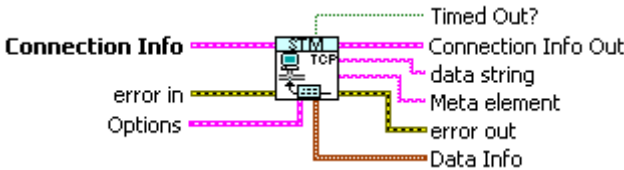


图4.41 STM Read Message vi

STM Write Metadata vi

使用这个VI将元数据信息发动到远程计算机上。为正确的解释信息，发送和接受端的元数据必须保持一致。在服务器上维护元数据，然后将其发送给客户端，这样就可以避免维护每个计算机上的元数据副本。

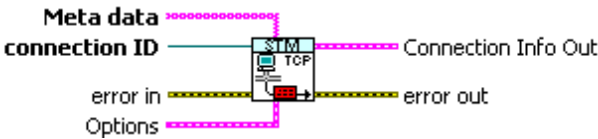


图4.42 STM Write Metadata vi

STM Read Metadata vi

使用这个VI接收远程计算机传来的元数据信息。这个VI读取和解压元数据数组，这些元数据数组能被传送的Read和Write VI

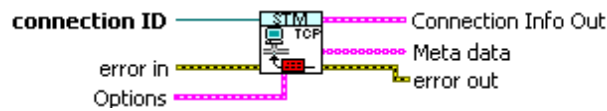


图4.43 STM Read Metadata vi

使用STM应用程序接口来发送数据

图 4.44 展示了一个使用STM API的数据服务器的例子。可以看到一旦连接完成，这个程序就会将元数据发送到一个远程计算机上。这个例子写入两个数值：循环计数器和一个双精度浮点型数组。元数据包含对这两个变量的描述。

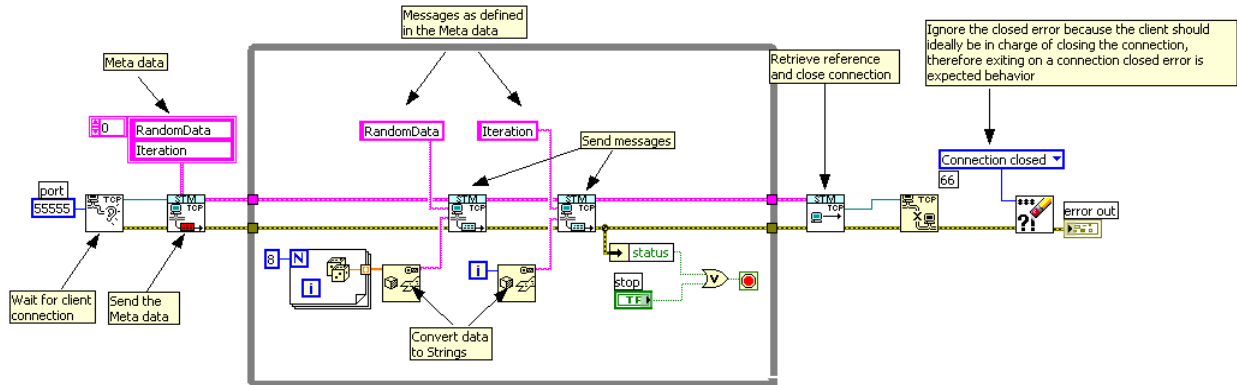


图4.44 基础STM服务器例子

使用STM API来接收数据

接收数据也非常简单。当与服务器建立连接之后程序就会等待元数据。然后程序使用STM Read Metadata vi等待即将到达的信息。当接收到信息之后，程序就会根据元数据名来转换数据并将数据分配给本地值。

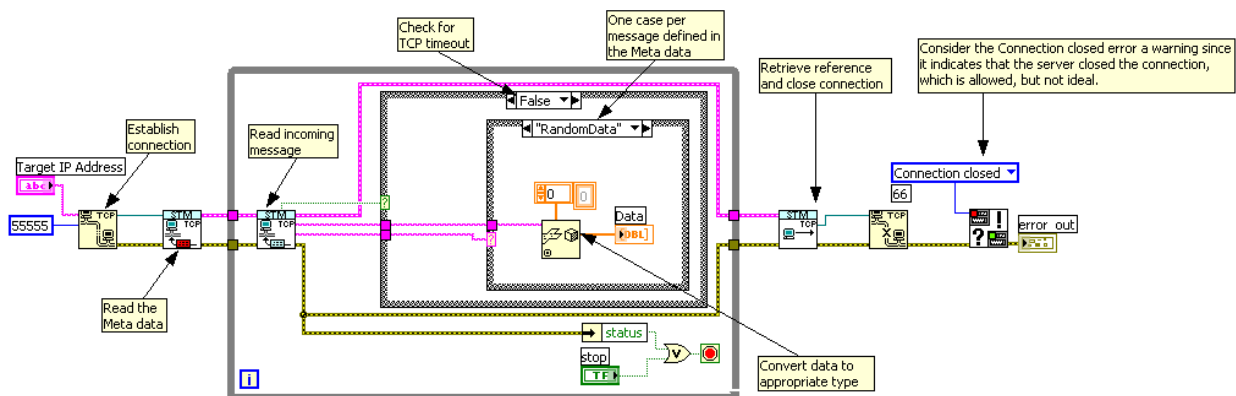


图4.45基础STM客户端例子

对于大多数的Lab VIEW和Lab VIEW之间的交流，NI推荐使用网络发布的共享变量；对Lab VIEW与其他系统之间的交流，NI推荐使用更高级的通讯协议，比如Modbus TCP。当然Lab VIEW也提供灵活的方式来创建自己的通讯协议比如STM协议。

Compact RIO的串行通讯

一个超过40年的标准，RS232（也就是EIA232或者TIA232）端口被应用于所有的Compact RIO系统以及各种工业控制器、可编程式逻辑控制器和设备上。RS232实际上是一个低成本的通讯标准，因为它并不复杂且只需要较低的加工和开销以及适度的宽带。然而在PC领域里，它正在退出应用，因为在这个领域里新的标准比如USB、IEEE1394和PCI Express更加实用。在工业领域里，RS232因为其上述的优点，现在仍然很实用。

RS232说明书包含了硬件的执行、低电平定时以及字节传送，但是它没有定义固定通讯规格。但是它考虑了基本字节读与写的简易执行以及不同设备间的复杂功能。大量的协议是使用基于字节信息的RS232比如Modbus RTU/ASCII、DNP3和IEC-60870-5。

这部分包含了怎么为NI Compact RIO实时控制器的内置串行端口编制低电平的通讯。实时控制器使用了NI-VISA API

RS232技术介绍

RS232是一个单点通讯标准，即只有两个设备之间可以彼此连接。该标准定了许多信号线，这些信号线被用在不同应用程序的多个方面。所有的串行设备都采用数据发送线（TXD）、接收线(RXD)，和地线。其他的通讯线包括流量控制线、请求发送和允许发送线，这些线有时候被用来改进设备之间的同步性。另外当接受端和发送端不能同步时，这些线被用来组织数据的流失。其他的线通常与调制解调器的主机一起使用。在工业应用程序里除了与无线调制解调器一起使用外，通常并不使用这些线。

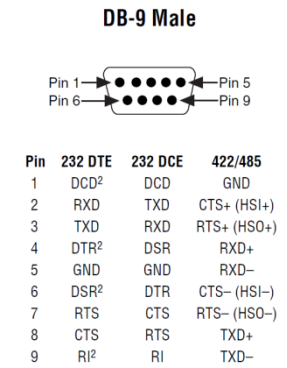


图4.46 D-Sub 9针连接器引线

作为一个单端总线，对短距离（10m一下）通讯来说RS232是非常理想的。屏蔽线降低了长距离上的噪声。对于长距离和速率来说，RS485更加合适，因为它使用不同的信号来降低噪声。不像总线比如USB和IEEE1394，RS232没有设计电源终端设备。

Compact RIO控制器有一个RS232端口，它是一个标准的D-Sub9针插头连接器（即DE-9和DB-9）。当控制台出门开关已被使用或者与RS232设备在应用程序里进行通讯时，可以使用这个端口来监测从Compact RIO控制器传来的诊断信息。

RS232接线和线缆

RS232端口具有两个标准插脚引线。可以将RS232设备分为数据终端设备（DTE），就像控制器或者主机，和数据通讯设备（DCE），就像从属机器。DCE端口具有反向的端口，这样DCE输入插脚就可以和DTE输出插脚相互链接。Compact RIO控制器上的串行端口是一个DTE端口；GPS、条形码扫描器、调制解调器和打印机上面的端口是DCE端口。

当连接Compact RIO DTE RS232端口和一个普通设备的DCE RS232端口时，可以使用一个定期直通电缆。当链接两个DTE或者DCE设备时，比如将Compact RIO控制器连接到PC，需要使用零调制调节器电缆。传送和接受的信号在电缆内交换。

设备1	设备2	缆线类型
DTE	DTE	零调制调解器电缆
DTE	DCE	直通电缆
DCE	DTE	直通电缆
DCE	DCE	零调制调解器电缆

图4.47 选择不同的电缆来连接不同的RS232端口

回环测试

RS-232 Loopback

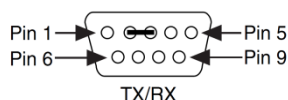


图4.48 RS232回环连线

回环测试是一个检验串口功能端口并能诊断与维修故障的通用技术。在基层所需要做的就是连接TXD和PXD插脚。使用一个标准的RS232直流或回环线和一个小弯曲别针来缩短插脚2和。使用回环线之后，读取函数就可以读取从读取函数发送的字节。可以检查这些软件、串行端口设置和驱动是否在正常工作。

Lab VIEW的串行通讯

本地Compact RIO RS232串行端口被直接连接到Compact RIO实时处理器上，因此必须编制程序来访问Lab VIEW实时VI上的串行端口。使用NI-VISA函数来发送和接受串行端口的数据。

NI-VISA是一个为字节级接口通讯提供单接口的驱动。字节级接口包括RS232、RS485、GPIB等。使用NI-VISA函数编写的程序可以在任何具有串行端口和NI-VISA的机器上运行。这就意味着可以在具有Lab VIEW的Windows机器上写入和测试串行VI，然后在Compact RIO的实时控制器上使用同一个程序。然后就可以直接使用位于ni.com/dnet的已存在的一起驱动程序。

为了开始运行串行端口，在函数选项板上找出虚拟仪器软件构架（VISA），它位于数据通讯》协议》串行。

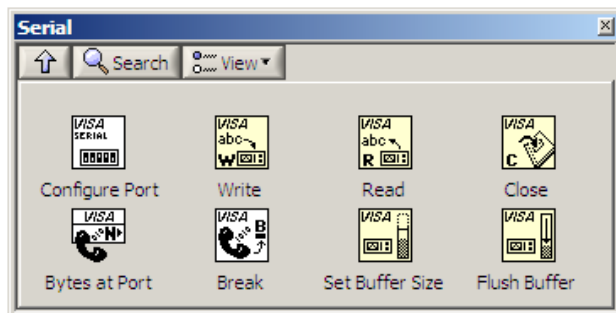


图4.48 VISA函数

对于大多数简单仪器应用程序，只需要两个VISA函数，VISA写和VISA读。从Lab VIEW\examples\instr\smp1ser1.llb中的基础串行Write和Read VI例子中能够学会怎么使用VISA函数。

能够从设备中读取信息之前，大部分的设备要求音信以命令或者疑问的形式发送。因此VISA Write函数之后通常跟随一个Read函数。因为串行通信需要配置额外的参数比如波特率，因此必须开始串行端口与VISA Configure Serial Port.VI的通信。VISA Configure Serial Port.VI通过VISA源名称来初始化端口特征。

必须注意到VISA源名称涉及到了目标机器上的资源。

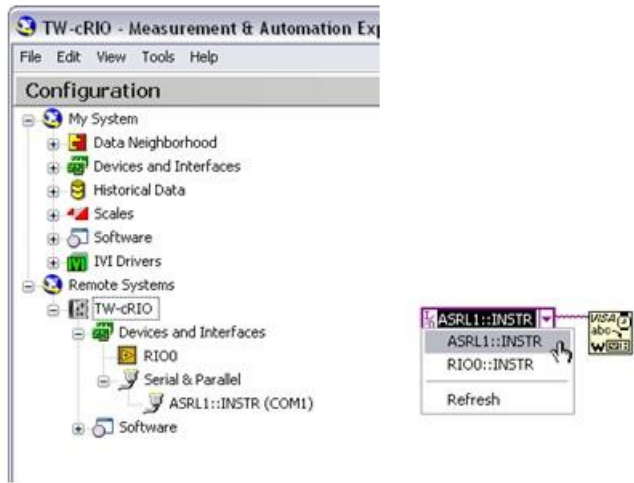


图4.49 可以从Lab VIEW资源控制菜单或测量与自动化浏览器上直接浏览VISA源名称

检查VI的左下角来决定VI与哪一个目标连接。对于Compact RIO，使用COM1来使用内建端口。Compact RIO 控制器上的 COM1 是 ASRL1: :INSTR. 如果连接了Compact RIO，那么就可以从Lab VIEW资源控制菜单或测量与自动化浏览器上直接浏览VISA源名称（可下载的功能用来配置所有的NI设备）。

为串行通信设置超时值。比特率、数据位、奇偶校验和流控制设定了那些特殊串行端口参数。错误输入和错误输出簇保证了VI的错误条件不发生变化。

当端口以字节水平工作，那么用于读和写函数的端口就是字符串。在内存里，字符串是字节的一个简单集合，这个集合包含字节的长度。这就使得程序使用多字节工作时变得更加简单。

Lab VIEW里的字符串函数提供的工具可以用来分解、操作和结合从串行端口接收的数据。也可以将字符串数据转换成一个字节数组数据。因为串行操作处理边长度的字符串，所以这些任务是非确定的。另外，从性质上说，串行端口是异步的并且它没有任何机制能保证实时得传送数据。当编写串行应用程序时，使控制程序的通讯保存在不同的循环里，这样就可以维持控制循环的决权。

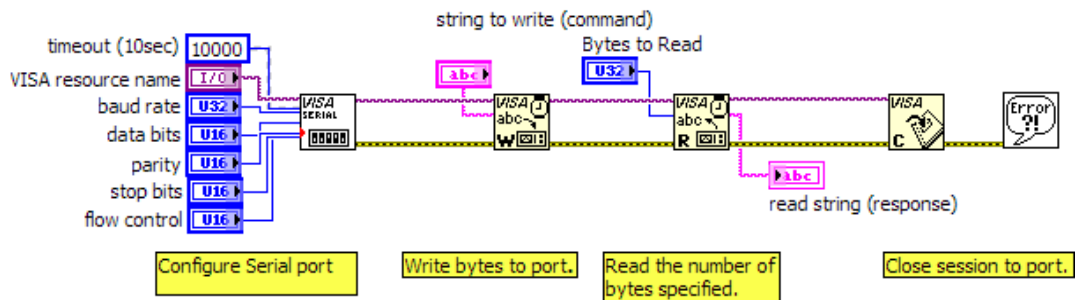


图4.50 简单的串行通讯例子

VISA Read函数会一直等待直到串行端口接收到了请求的字节或者超时。对这个应用程序来说，它允许这个行为，但是一些程序会接收不同长度的数据。在这种情况下，就应该只读取那些能够被读取的字节。可以使用串行端口属性的字节属性来完成这个操作，在串行函数板上可以找到这个属性。

可以使用属性节点来存取串行端口的参数、变量以及状态。如果打开VISA Configure Serial Port VI并且配置程序，就可以看到这个VI会简单的调用很多属性节点来配置端口。可以使用这些接口来存取高级串行端口特征，这些特征包含辅助数据线、等待读/写的数据和内存缓冲的大小。图4.50展示的例子中使用属性节点在端口读取之前来检查端口中可用的字节。

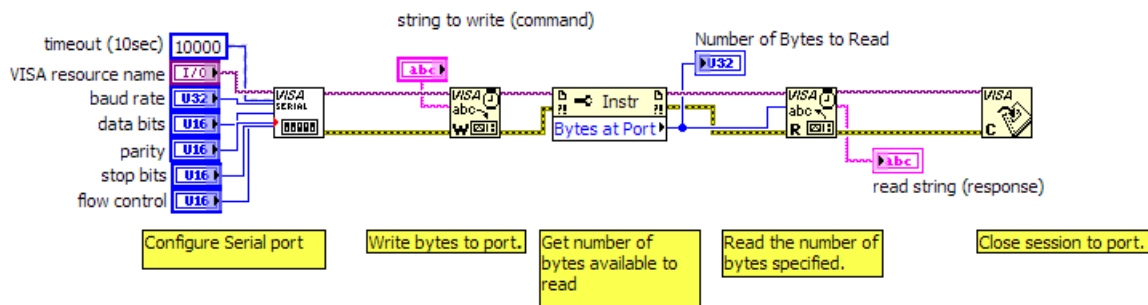


图4.51使用VISA属性节点来读取指定数量的字节

通常来说，设备的工作范围可以从简单的周期性读取广播字节比如GPS到复杂的命令构架。可以在ni.com/idnet 上找到一些设备或者仪器的驱动程序，还可以进行简单的开发。因为这些驱动程序使用NI-VISA函数，所以必须使用Compact RIO的板载串行端口。

仪器驱动网络

为了方便加速开发，NI公司和主要的测量与控制供应商提供了一个驱动函数库，这个驱动函数库里包含了多大7000个设备的驱动程序。仪器驱动程序是一组控制可编程仪器的软件程序。每个程序都对应一个操作，比如配置、读取、写入和触发仪器。仪器驱动程序简化了仪器的控制并减少了程序开发的时间，因为使用仪器驱动就可以不用学习每个设备的编程协议。

在哪里寻找仪器驱动并下载他们呢？

可以通过两种方式来寻找并下载仪器驱动程序。如果使用LabVIEW8.0或者更高的版本，最简单的方式就是使用NI Instrument Driver Finder。如果使用较早版本的Lab VIEW，则可以使用Instrument Driver Network（ni.com/idnet）。

使用NI Instrument Driver Finder寻找和下载并安装仪器的Lab VIEW即插即用驱动程序。选择Tools》Instrumentation》Find Instrument Drivers来开启NI Instrument Driver Finder。这个工具搜索认证网络来寻找指定的一起驱动。图4.52展示了怎么从Lab VIEW启动NI Instrument Driver Finder。

可以为特定的仪器使用一个仪器驱动。然而Lab VIEW即插即用仪器驱动分布在程序框图的源代码中，因此可以为特定的应用程序自定义即插即用一起驱动。编程将仪器驱动VI连接到程序框图，这样就创建了仪器控制程序和系统。

Lab VIEW中串行通讯的例子



提供例子的Lab VIEW程序代码

现在修改原始的PID(比例-积分-微分控制器)例子，这样就可以从串行仪器上取回温度值，而不用从内置的热电偶模块上得到它。在这个例子中从Lake Shore Cryotronics 211上读取温度值。

在认证网络上可以搜索并下载到仪器驱动。也可以在Lab VIEW上直接搜索到驱动程序。

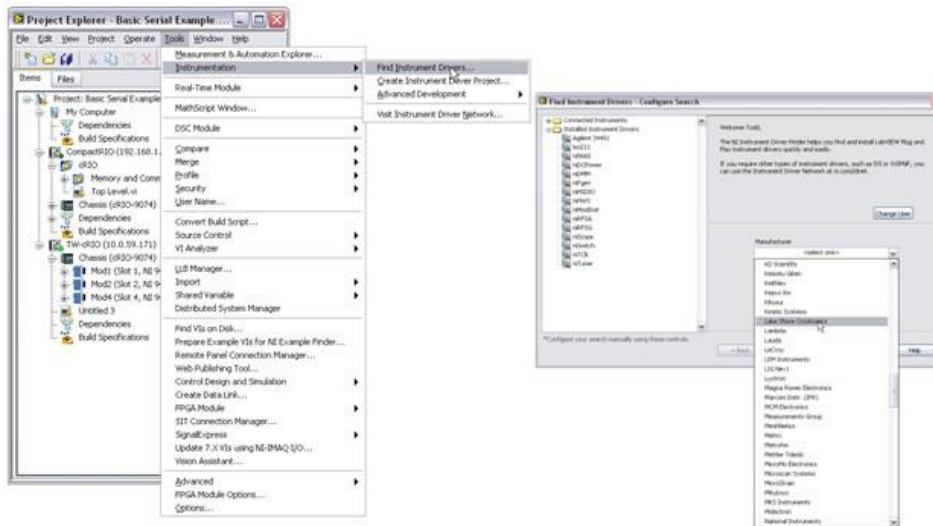


图4.52 使用Lab VIEW可以直接搜索并下载多达7000个设备的通讯驱动

这个功能自动下载驱动并将它们保存在Lab VIEW安装目录的instr.lib文件夹里。Lab VIEW运行时，保存在这个文件夹里的驱动会自动出现在函数板上。因此在Lab VIEW里，可以在仪器驱动函数板上看到LSC1211温度监视器的新标识。

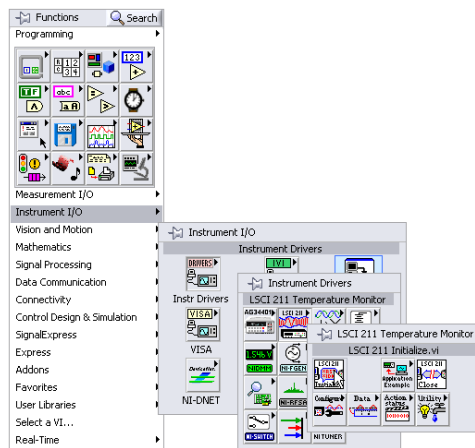


图4.53 可以从仪器I/O函数板上浏览仪器驱动

接下来修改PID应用程序，这样热电偶1就变成了一个单进程的可以实时先进先出的共享变量。在初始化程序里，给热电偶1设置一个默认值。

最后给串行通讯任务增加第二个循环。进入循环之前，为设备初始话串行通讯，在 循环里读取稳定值并将其写入到记忆表中（单进程共享变量）。停止循环时关闭串行通讯。

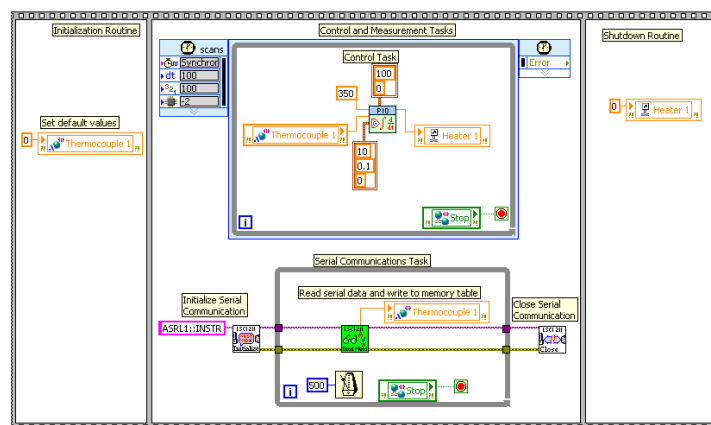


图4.54 完整的应用程序，从串行设备中读取温度值，并将其发送到高优先级的控制循环中

Compact RIO的RS232和RS422/RS485 4端口 NI C系列模块

如果应用程序需要更高的性能或者额外的RS232端口或者RS485/RS422端口，那么就可以为Compact RIO使用NI9870和NI9871串行接口，这样就能增加更多的端口。使用Lab VIEW FPGA就可以从FPGA中直接访问这些办卡。在下面的章节里会详细的技术FPGA编程技术。

与PLCs（可编程时逻辑控制器）或其他工业网络设备的通讯

应用程序通常需要将NI PAC（可编程自动化控制器）比如Compact RIO整合到一个工业系统或者与其他的工业设备整合到一起。这些系统通常由传统的PLC、PAC，或者各种特殊设备比如电机控制器、传感器和HMI（人机界面）组成。这些设备可以提供大范围的通讯选性，这些选性可以从简单的RS232到专门的高速度工业网络。

有三种通用的方式可以将Compact RIO链接到工业系统或者PLC。使用标准的模拟或者数字I/O可以将Compact RIO直接连接到PLC。这个方法非常简单但是近使用于很小的系统。对于大型的系统，可以使用工业通讯协议在Compact RIO和PLC之间通讯。对于大型SCADA（监控与数据采集系统）应用程序，OPC是一个很好的工具。OPC可以应用到高速通道，但是它使用Windows技术，因此需要Windows PC来执行通讯。

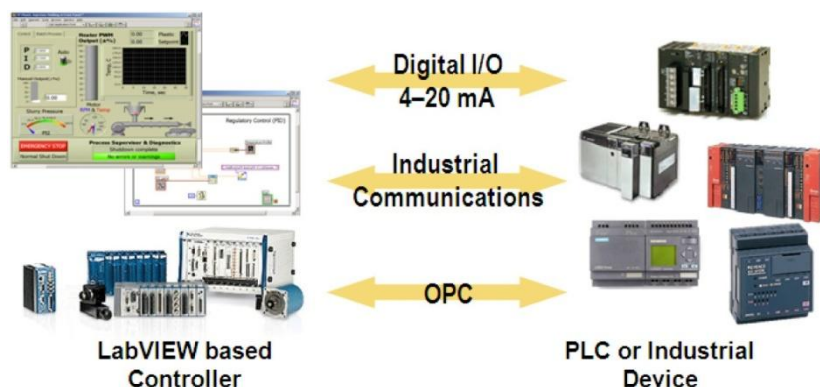


图4.55 链接Compact RIO控制器与工业设备的三种方式

工业通讯协议

除了使用标注I/O之外，链接Compact RIO与其他工业设备的最常用的方式就是使用工业通讯协议。大多数的协议使用物理层，比如RS232、RS485、CAN或者以太网。就像前面所讨论的那样，总线比如RS232和以太网只为简单的通讯提供物理层，缺乏为高等级的通讯提供任何预定义的方式，且仅能满足使用者接受或者发送单独的字节或者信息。当工作中面临大量工业数据，处理这些数据并将它们快速转化为相关的控制数据就变得冗长而繁琐。工业通讯协议为如何交流数据提供了一个框架。虽然它们运行更复杂的协议栈，但从根本上来说它们与上文讨论的仪器驱动还是很相似的。

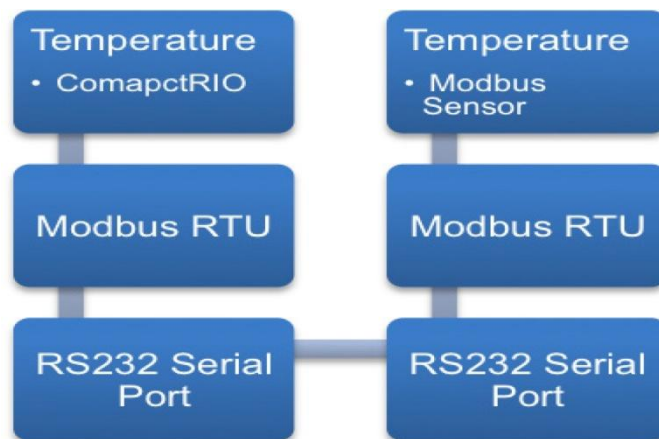


图4.56典型工业协议的执行过程

工业协议通常携带一个低级总线像RS232或者TCP/IP，并且为顶层的通讯增加预定义技术。这与HTTP非常相似。HTTP是一个传送网上内容的协议，它位于TCP/IP的顶部。可以编写自己的HTTP客户端就像网络浏览器或火狐浏览器，这需要消耗很大的精力。如果不进行大量的测试和检验就不能保证自定义的HTTP客户端能够连接到所有的服务器。同理，支持工业网络标准的设备和控制器在程序层面上时非常容易整合的，并且能够从最终用户那里提取底层的通讯细节。这样设计者就可以只关注应用程序。

因为Compact RIO具有板载串行和以太网端口，所以它支持很多协议像Mod bus TCP、Mod bus Serial、Ethernet-IP和其他协议。同时它也具有可用的插入式模块来与更多的通用工业协议通讯，比如FROFIBUS、CAN open等。

如果本地通讯不能使用，那么就可以选择使用网关。网关是协议转换服务器，可以转换不同的工业协议。比如想要连接到CC-Link网络，就可以使用网关。网关可以在PAC端使用Mod bus TCP，在另一端将其转换成CC-Link。

Mod bus 通讯

Mod bus是现在使用最广泛的工业协议。这个协议诞生于1979年，它支持串行和以太网物理层。Mod bus是设备之间进行客户机/服务器通讯的应用程序信息协议，这些设备通过总线或者网络连接。通过使用异步串行传输与触摸屏、PLC和网关之间进行通讯来执行Mod bus，这个过程也支持其他类型的工业总线。Mod bus可以与Compact RIO以及其板载串行或者以太网板一起使用。必须注意到Compact RIO板载串行端口使用RS232，而一些Mod bus却使用RS485电层。在这种情况下就需要使用通用的RS485到RS232适配器。

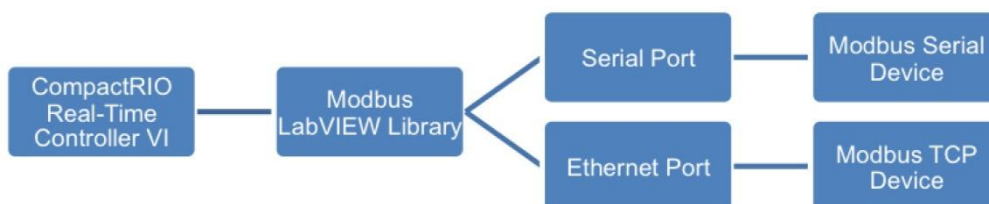


图4.57 与Mod bus设备通讯的软件构架

Mod bus串行协议是一个基于主/从构架开发的协议。每个从属设备都分配了一个从1到247的地址。在任何时间，只有主机与总线连接。除非主机请求否则重属设备不会传送信息，并且重属设备之间不能进行通讯。

通过在位于重属设备的存储器上读取和写入，信息就可以在主机和重属设备之间进行传递。Mod bus说明里展示了4个不同存储器的使用，每个存储器都可以容纳65,536个项目，他们通过注册类型和读-写存取进行区别。在Lab VIEW执行过程中，存储器不会重叠。

表格	数据类型	存取类型	注释
离散输入	单比特	只读	主机只能读取，自由重属机器能够改变其存储器

			的值
线圈	单比特	只写	主机和重属设备都可以从存储器中读取和写入。
输入存储器	16位字	只读	主机只能读取，自由重属机器能够改变其存储器的值
保持存储器	16位字	只写	主机和重属设备都可以从存储器中读取和写入。

表4.2 通过向位于重属设备的存储器中写书和读取来在主机和重属设备之间传递信息

可以在ni.com上看到Mod bus的详细介绍。也可以下载免费的Mod bus Lab VIEW Library。在串行通信方面，Mod bus Lab VIEW Library使用与NI-VISA函数的使用是相似的。在Function》User Libraries》NI Mod bus上可以找到相应的函数板。在函数板的第三行和第四行可以找到相关的例子。

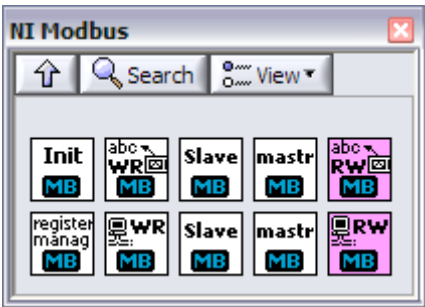


图 4.58 Lab VIEW为主机和重属设备的通讯提供了Mod bus函数库

在使用Mod bus编程之前，首先要根据其书名设置一些Mod bus设备的重要参数：

1. 主机或者重属设备？如果设备是重属设备，那么就将Compact RIO配置为主机。这是最 常用的配置方式。同样连接Mod bus主机（比如另一个PLC）时，要求将Compact RIO配置为重属设备。
2. 串行或者以太网？Mod bus以太网设备有时候又叫做“Mod bus TCP” 设备
3. 对于Mod bus串行：
 - RS232或者RS485?许多设备使用RS232，但是有些设备长距离传输时使用RS485总线。RS232设备可以直接插入到Compact RIO系统，但是RS485设备需要一个RS232- RS485转换。
 - RTU还是ASC II模型？RTU/ASCII涉及到了串行总线上的数据格式。RTU数据是原始二进制数据，而ASCII是一种可以认为读取的数据。Mod bus VI需要这些参数
4. 存取器地址是什么？每一个Mod bus设备都有自己的一个到存取器、线圈和离散输入的I/O映射，这些映射以数字地址的形式给出。按照Mod bus习惯，存储器得之一般是一个小于存储器名称的地址。这与一个数组的第一元素是0是相似的。Mod bus Lab VIEW Library需要存储器的地址而不是名字。

Mod bus范例

这个例子显示了怎么在最常用的Mod bus实施中，将Compact RIO作为RS232串行端口上的一个Mod bus RTU主机。为了简单没有链接设置值，所以程序以它的初始值运行。

- 波特率：9600，没有奇偶校验、没有数据流控制、超时值为10s
- RTU数据格式
- 重属设备地址是0

如果设备使用不同的参数，确保链接了所有受影响的VI

虽然例子展示的是在串行端口上使用Mod bus，但是通过将VI替换为函数板上相应的以太网函数，就可以很容易的将这个例子改为使用TCP/Ethernet。

运行时，这个例子从Mod bus设备中读取大部分最新的温度值，缩放数据然后将他们放置在主要的内存列表中。然后读取最新的输出值，将他们转化为整型数据，并且更行重属设备上的值。为了将这个例子整合到控制结构中，将这个循环作为另一个并行的“驱动”任务来运行。使用可以先进先出的单进程共享变量将Mod bus设备的输出和输入值保存在一个记忆表中。

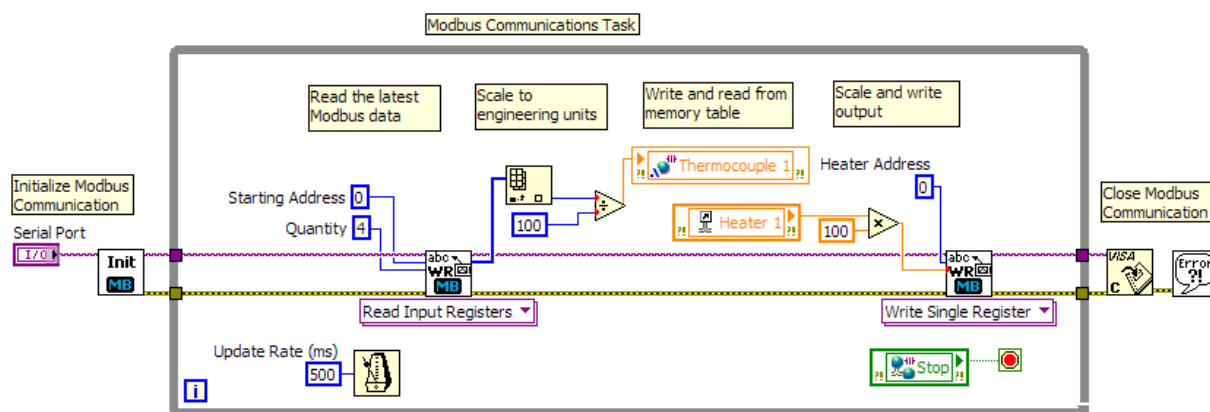


图4.59 一个Mod bus通讯的Lab VIEW例子

首先，选择串行端口。Compact RIO的端口是ASRL1::INSTR。如果目标VI连接到Compact RIO，那么该端口应该自动填充。然后Mod bus主循环开始运行。这个例子展示了一个简单的输入寄存器的读出/写入。寄存器用来储存多值数据和16位的整数。通常，输入寄存器读取一批数据，因此要指定开始地址，同时也要按次序指定要被读取的地址。这样用Read VI进行一次调用，就可以读取所有的数据。

比如，从数组中检索第一次返回的值，将其标定为工程单位，然后纺织袋记忆表中。标定根据传感器、校准和其他因素由使用者决定。通常使用者将这个标定功能放置在一个子VI。

调用Mod bus Write将最新的输出数据写入到Mod bus网络。数据必须再次被标定为Mod bus使用的整数形式，然后将其发布到合适的主机地址。这个例子展示了一个单数的寄存器写入。当然它也可以一次更新很多数值。

这个例子中循环更新速率是500ms，当然可以将其调整到终端设备所期望的更新速率。一定要记住最大的循环速率取决于读/写的寄存器/线圈的数量、Mod bus连接的波特率和终端设备的能力。从总体上来说，Mod bus/TCP设备的更新速率能够比Mod bus串行设备的更新速率大很多。也要考虑当循环速率达到极限速率10ms，可以使用Compact RIO的极限CPU时钟，减少其他任务的处理时间。

Ethernet/IP

另一个非常通用的工业协议就是Ethernet/IP。Ethernet/IP通常被用在现在的Rockwell PLCs上，且使用标准的Ethernet缆线与工业I/O进行通讯。NI在ni.com/labs上有一个可用的库，它允许Compact RIO凭借Explicit Messaging直接从PLC读取或写入标签来进行通讯或者与一个完整的Ethernet/IP Class 1 Adapter（重属机）进行通讯。当库是实验室产品时，那么它是从一个经过检验的Ethernet/IP协议中创建的，并且经过了广泛的测试。通过运行安装程序来安装驱动。

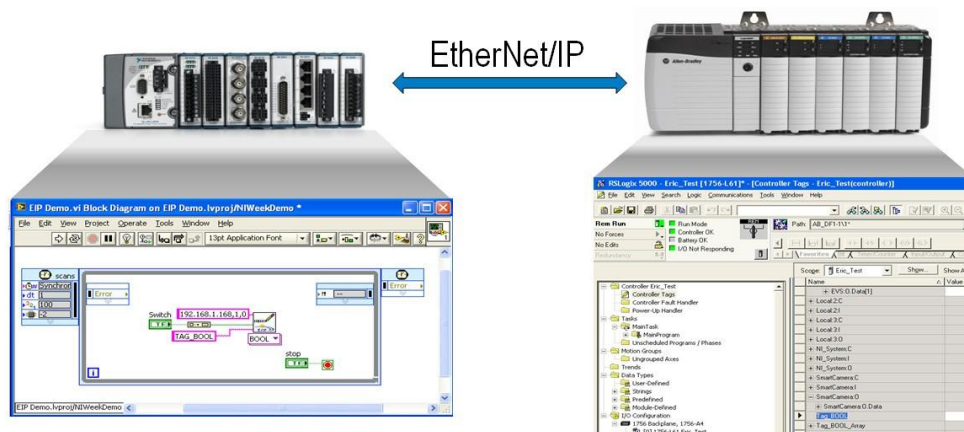


图4.60使用Ethernet/IP，可以从Compact RIO的Rockwell PLC上直接读取或者写入标签

使用与Mod bus库相同的技术，Ethernet/IP驱动被整合到控制结构中。创建另一个并行循环作为“驱动”任务来工作。使用可以先进先出的单进程共享变量来将Ethernet/IP网络上的数据传送到记忆表里。

OPC

OPC (或称为用于过程控制的或OLE)，是被用在许多应用程序的通用协议。这些应用程序具有高通道数和较低的更新速率，这在工业进程中是非常普遍的，比如石油、煤气和制药领域。OPC被用来连接HMI与控制器以及SCADA与控制器，但是它不能用于控制器之间的通讯。一个OPC最普遍的使用方式，就是为了HMI与SCADA应用程序将Compact RIO控制器整合到第三方系统中。

OPC说明书是大型的标准的集合，这个标准跨越了许多领域的应用程序。这部分主要讨论OPC数据存取，说明书的主要部分是介绍怎么使用通用的Windows技术进行数据传输。Lab VIEW的网络共享变量为OPC提供了一个网关。

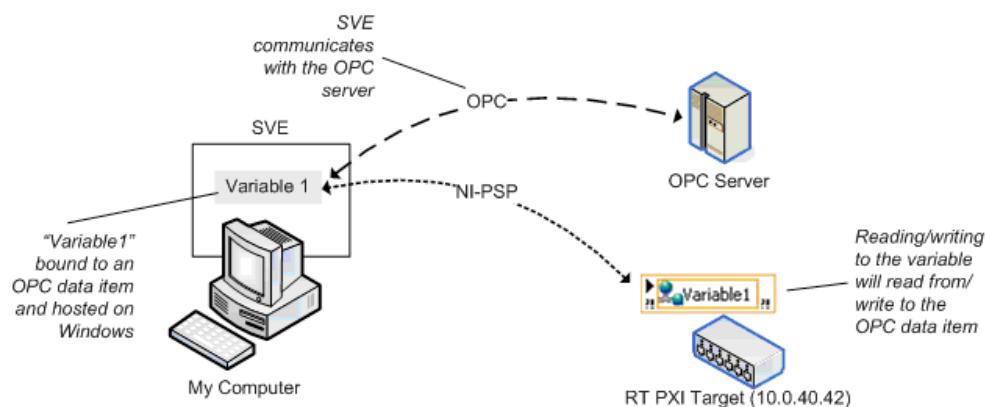


图4.61网络发布的共享变量为OPC提供了一个网关

OPC只是一个Windows技术，所以Compact RIO不能直接使用OPC协议进行通讯。然而Windows PC可以通过PSP与Compact RIO进行通讯，并能通过OPC来解释和发布信息。当运行Windows机器时，NI共享变量引擎作为OPC工作。服务器自动执行翻译和发布。这就因为这其他OPC客户端比如第三方SCADA数据包或者进程控制软件可以从Compact RIO设备中存取或者检索数据。

通过OPC从Compact RIO上发布数据

下面的例子描述了怎么将Windows共享变量引擎作为OPC服务器使用来使数据变得可用。这个例子假设已经通过Compact RIO系统的PSP协议(网络发布的共享变量)将数据发布到网络上。在这个例子中网络共享变量被命名为“SV_PID_SetPoint”。

1. Windows PC也拥有网络发布的共享变量。任何基于网络共享变量的Windows通过默认也被作为一个OPC项目发布到网上。

2. Windows变量被“捆绑”到从Compact RIO系统中传来的数据，这样就会使 Windows变量 引擎订阅并接受从 Compact RIO系统中传来的数据并且更新OPC项目。
3. 其他的OPC客户端也连接到Windows PC，并使用数据进行显示、人机交互等。
4. Compact RIO使用Compact RIO控制器上的网络发布的共享变量将数据发布到网络上。

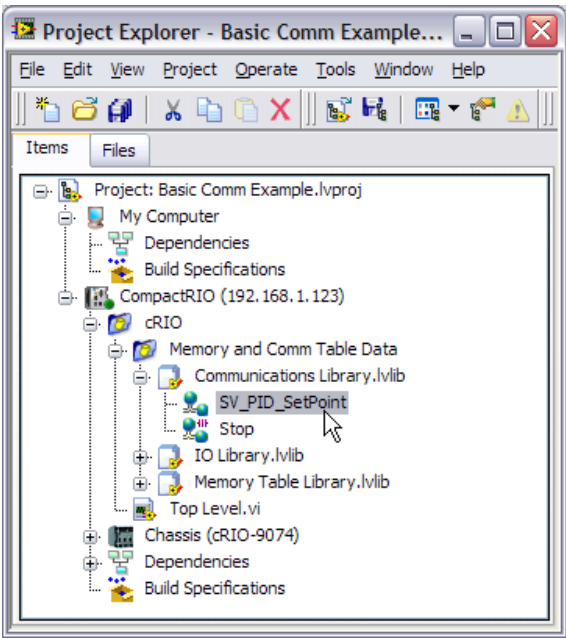


图4.62CompactRIO系统拥有的网络发布的共享变量

5. Windows PC也拥有网络发布的共享变量。任何基于网络共享变量的Windows 通过默认 也被作为一个OPC项目发布到网上。在Lab VIEW项目里，Windows共享变量位于My Computer。右击My Computer并选择New》Variable。

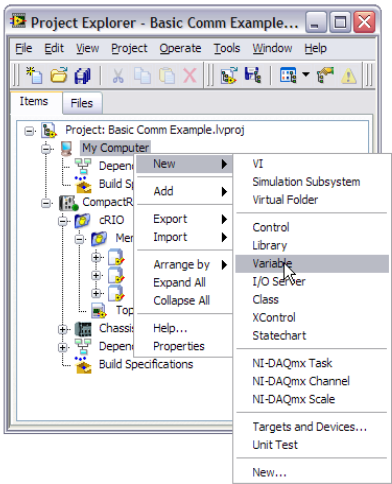


图4.63 Windows拥有的网络发布的共享变量会通过OPC自动发布

6. Windows变量被“别名化”为从Compact RIO传来的数据，这样就会使Windows变量引擎订阅并接受从Compact RIO系统中传来的数据并且更新OPC项目。在共享变量属性窗口，给变量一个可以使用的名称，比如“SV_PID_Set Point _OPC”。检查 Enable Aliasing复选框，并点击浏览按钮来浏览Compact RIO上的SV_PID_ Set Point变量。

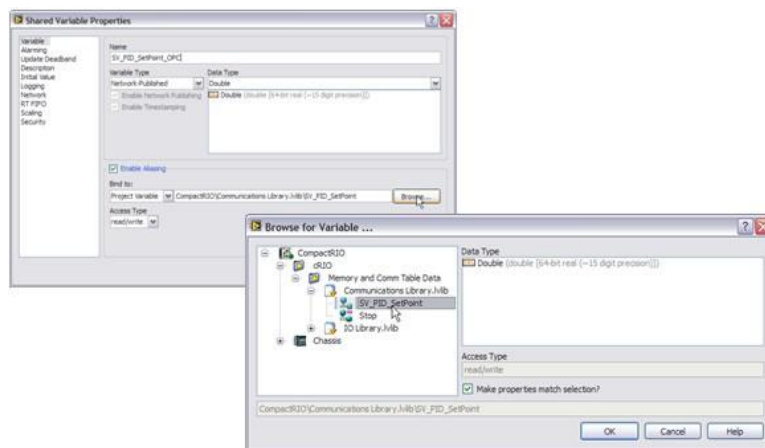


图4.64 别名化的网络发布的共享变量为Compact RIO和OPC提供了一个网关

7. 保存新建的库，这个库拥有一个可使用的名字，比如“OPC Library .lvlib”。
8. 通过右击库并选择配置，将变量配置到共享变量引擎。
9. 共享变量被作为OPC项目进行发布。打开OPC客户端比如NI Dstributed System Manager(Start»Programs»National Instruments»NI Distributed System Manager)就可以检查变量是否正确工作。

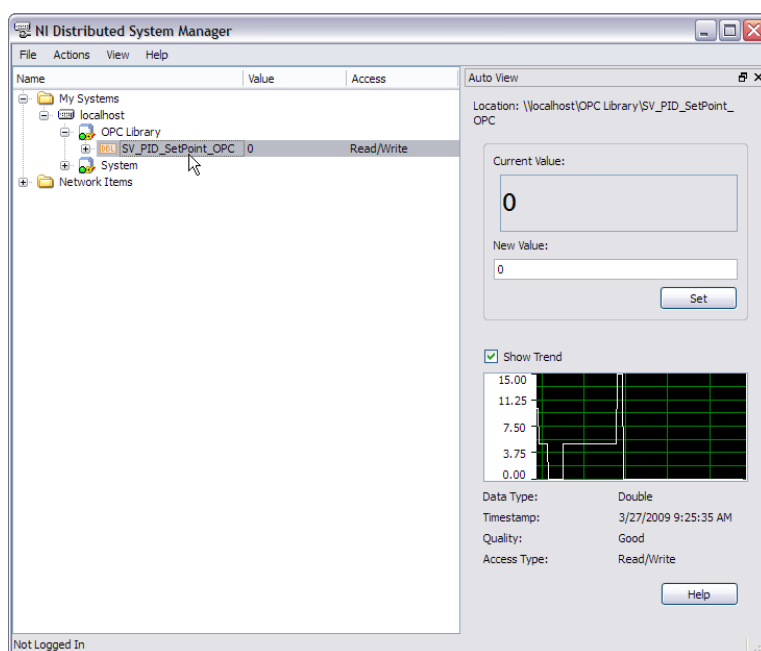


图4.65 OPC客户端，比如NI Dstributed System Manager，能够读取和写入OPC项目

10. 如果拥有一个OPC客户变量，比如 Lab VIEW DSC Module，也可以从客户端检查变量是否正确工作。NI 共享变量的OPC项目以National Instruments. Variable Engine.1服务器的名称发布。
11. 如果在共享变量引擎中配置了变量，那么重启服务器机器之后，共享变量将仍然保存在引擎中。

在ni.com可以找到关于共享变量引擎的更多信息。

第5章

添加I/O至Compact RIO系统

添加I/O至Compact RIO

典型Compact RIO系统的母板能容纳多达8个NI C Series模块，但有些控制应用需要更多的I/O通道或由主控制器分配的I/O。可以采用两种方法轻松扩展你的硬件系统：

- 以太网I/O
- 确定性以太网I/O

以太网I/O

使用以太网I/O扩展Compact RIO涉及到一个或多个Compact RIO系统共用相同的网络。主控制器负责通过使用两个不同系统的I/O，运行实时控制回路。而扩展控制器无法在其处理器上逻辑运行，且没有实施网络监管的简单逻辑，故仅为主控制器提供扩展或分布式I/O。编程人员可以使用已有的网络架构，如交换机和路由器。虽然全双工以太网路交换器消除了数据包冲突，但交换机仍会发生抖动，且通用以太网仅适用于无需确定性以太网的情况。如须同步局部I/O与扩展I/O，请参阅“确定性以太网I/O”部分了解更多信息。

使用另外的Compact RIO系统作为以太网I/O，适用于以下几种不同的架构：

1. 主机运行Lab VIEW代码的同时，为两个Compact RIO系统提供I/O和（或）附加进程
2. 主控制器和扩展控制器共同运行嵌入式real-time/FPGA逻辑并使用I/O别名或共享变量实现信息共享

步骤1. 构件扩展系统

添加另一Compact RIO系统作为以太网I/O的最简单方法，是将两个Compact RIO系统连接至相同网络。如何配置每个控制器及其IP地址，在该文档随后的“Compact RIO入门指南”部分有详细说明。



图 5.1. 添加另一Compact RIO系统作为以太网I/O的最简单方法，是将两个Compact RIO系统连接至相同网络

两个控制器均获得IP地址后，将它们添加至Lab VIEW项目中。选择扫描界面作为二者的编程模式。Lab VIEW Project Explorer窗口将在主系统和扩展系统中自动添加所有模块及I/O通道。

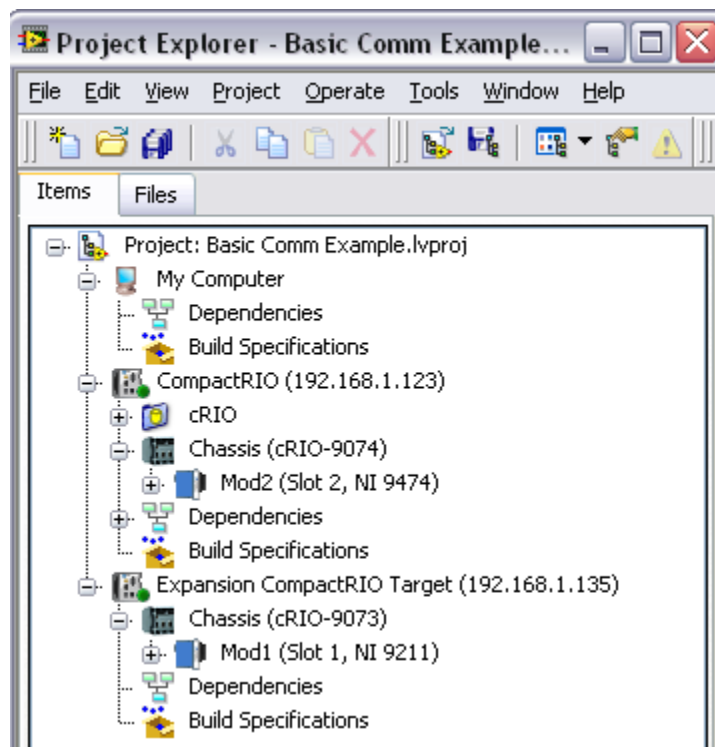


图5.2 Lab VIEW Project Explorer窗口显示Compact RIO系统由TCP/IP连接

扩展控制器的I/O由PSP协议自动发布。可以直接将I/O拖入主控制器框图来读写扩展控制器。

你可以通过改变网络发布时间，调整扩展机箱在网络上发送更新数据的周期。在控制器和选定的**Properties**右击。在Real-Time Compact RIO Properties窗口左边菜单的选择**扫描引擎**来读取这些周期。

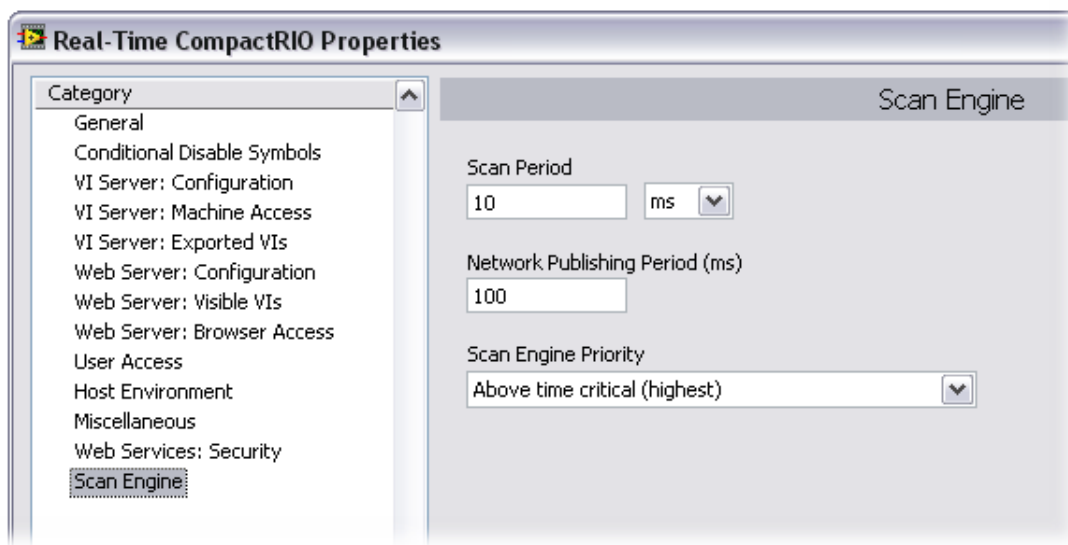


图5.3. 可以在Compact RIO控制器的属性中调整扫描周期及网络发布周期

断开以太网的注意事项

扩展控制器被用来运行嵌入式实时程序，因此并无网络监管。这意味着如果主控制器和扩展控制器的连接被破坏，输出结果将保留在前一状态。如不采纳该方式，则应为扩展控制器编写实时应用程序，以监测主控制器的跳动并将断开时的输出置于安全状态。

步骤2 添加I/O至主控制器的扫描

在主控制器上运行的程序中，读取I/O并将其添加至I/O扫描。前述部分详细介绍了从以太网等来源添加数据至I/O扫描。这里仅做简要

回顾。

异步非确定性I/O

因为通过标准以太网读取数据，这类I/O是非确定性的。对于这种使用情况，你应该创建一个定期while循环或低优先级的定时循环。由于这种循环不会被自定义I/O扫描任务中可能存在的高抖动I/O设备影响，这将使控制任务确定并可靠地运行。根据需要的刷新率设定任务定时。

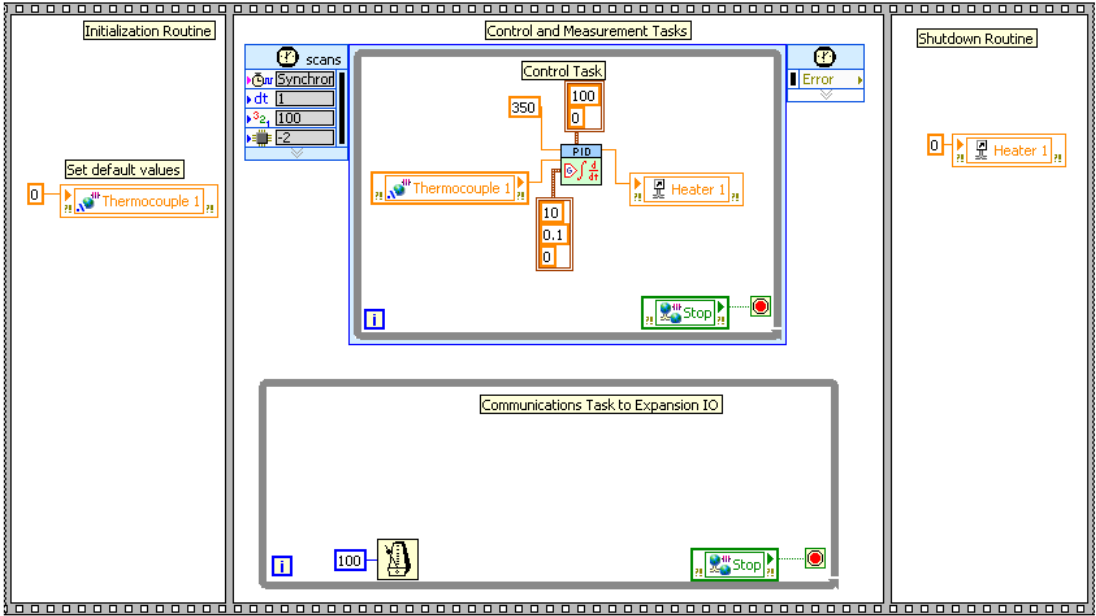


图5.4.添加循环或定时循环以处理I/O扩展扫描任务

步骤3 拷贝数据至内存

为访问整个应用在自定义I/O扫描任务中读入和写出的数据，使用实时FIFOs创建单进程共享变量，如“任务间的数据传递”部分所讨论。

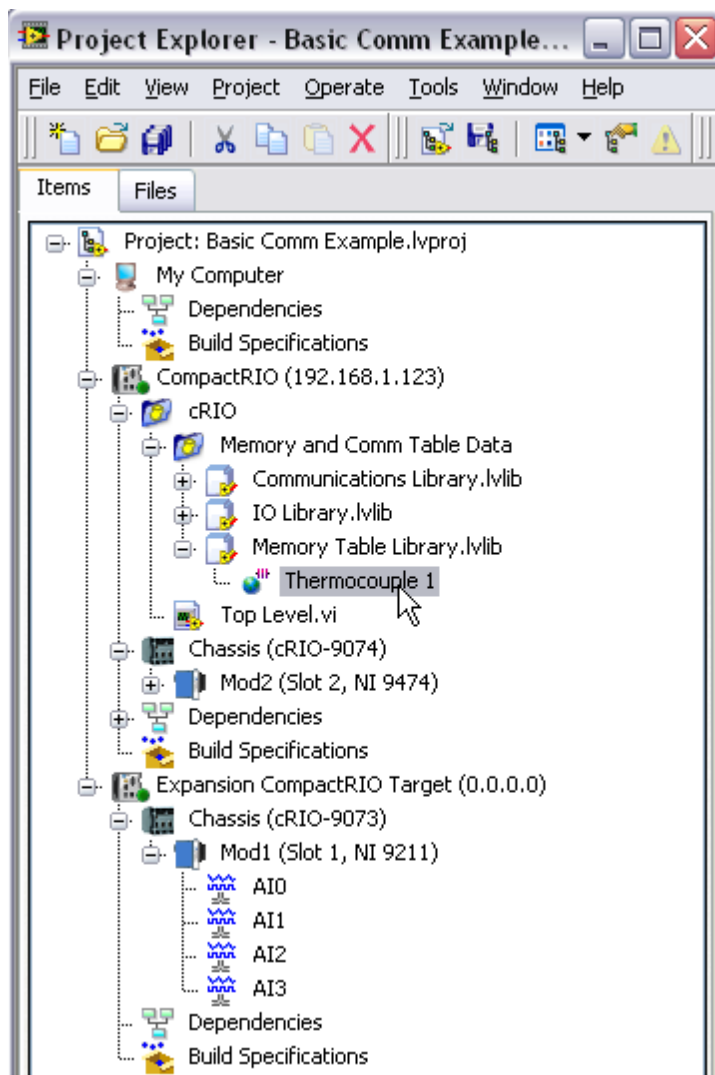


图5.5. 使用单进程共享变量与其它任务共享自定义I/O数据

贴士：对于高通道数系统，你可以使用Multiple Variable Editor，通过输出和输入.csv 的电子表格文件来合理化多I/O别名的创建。

在循环内，断开网络读取变量，检查警告或错误，同时将数据写入单进程共享变量。

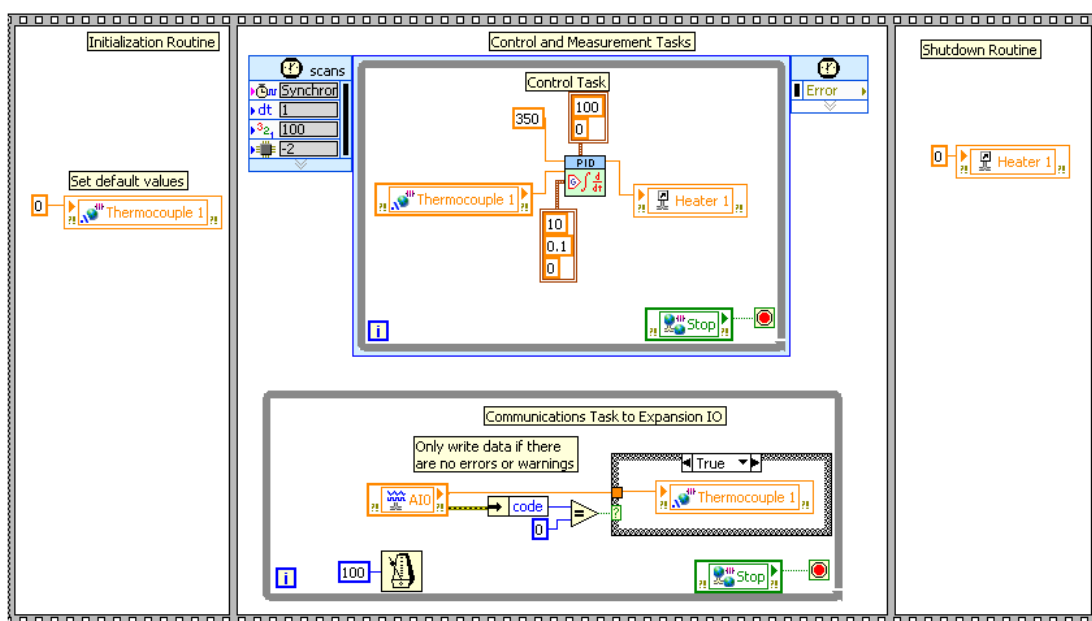


图5.6. 在通讯任务中使用单进程共享变量从扩展机箱重读取数据并写入内存

确定性以太网I/O

提供例子的Lab VIEW程序代码在某些应用中，主I/O及扩展I/O系统需要严格同步，进而要求所有输入输出同步刷新。使用确定性总线将允许主控制器获取扩展I/O刷新时间及准确的数据送达时间。你可以使用NI 9144扩展机箱方便地分配具有确定性以太网技术的Compact RIO。

NI 9144简介-确定性以太网机箱

坚固的NI 9144扩展机箱使用名为Ether CAT的确定性以太网协议来扩展Compact RIO系统。在主从式架构下，你可以使用任意带有两个以太网端口的Compact RIO控制器作为主设备，并将NI 9144作为子设备。NI 9144也带有两个端口，允许来自控制器的菊花链，以扩展时间先决的应用程序。



图5.7. Compact RIO硬件采用NI 9144确定性以太网机箱扩展时间先决系统

8插槽机箱支持全部用于虚拟、数字、动态及专业测量的C Series I/O 模块。你可以应用Lab VIEW Real-Time和Lab VIEW FPGA模块对NI 9144扩展机箱进行编程。对于简单扩展I/O，在Lab VIEW Real-Time扫描模式下编程可采用方便的开箱模式。扫描模式下的I/O变量可通过简单的拖放操作实现物理I/O值的即时接入，并提供监控系统性能的现场试验盘，执行高级故障排查。同时，它包含网络监管系统，如果它与主设备断开连接，则将扫描模式下的输出值设为默认状态（0V或关闭）。

在FPGA模式下编写分布式I/O为整体定制水平及应用的灵活性打开大门。通过嵌入决策制定功能，它将减少响应时间，迅速对环境作出反应而无需与主机交互。智能扩展I/O还能够通过在节点上执行内联分析、定制触发及信号处理，从控制器卸载进程。此外，使用Lab VIEW FPGA为领域专家塑造并实施其构想开辟了道路。更多关于Lab VIEW FPGA编程的细节，请参阅第6章，“通过Lab VIEW FPGA定制硬件”。



图5.8. NI 9144扩展机箱

步骤1. 安装确定性扩展机箱

为使Ether CAT支持Compact RIO控制器（以NI cRIO-9074为例），你必须在主机上为Ether CAT驱动安装NI-Industrial Communications。Ether CAT驱动适用的NI-Industrial Communications包含在CD内NI 9144机箱部分，并可从ni.com免费下载。



图5.9. 配置确定性以太网系统，连接cRIO-9074端口1至主机，连接端口2至NI 9144

将Compact RIO配置成确定性总线主控

1. 运行Measurement & Automation Explorer (MAX)并连接至Compact RIO控制器。
2. 在MAX界面，在Configuration面板的Remote **Systems**下扩展控制器。右击**Software**并选择**Add/Remove Software**。

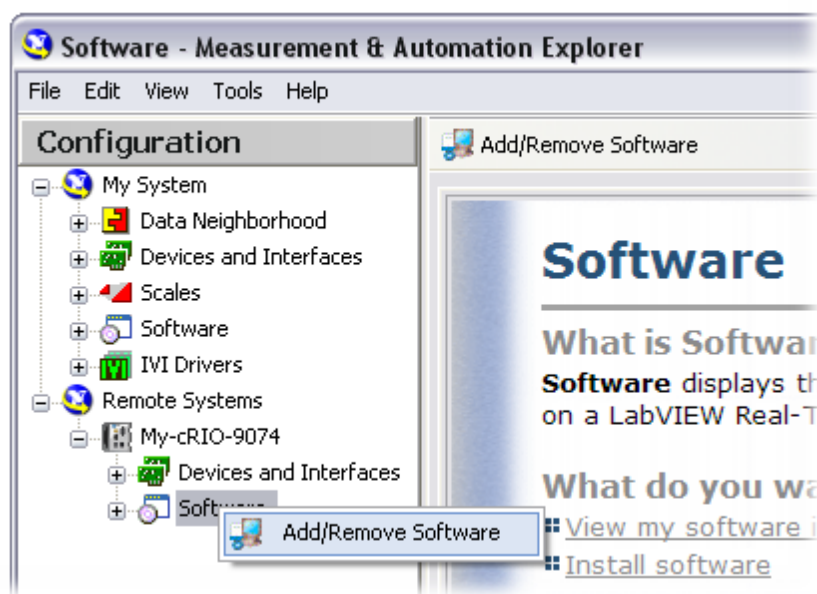


图5.10. 在Measurement & Automation Explorer下安装适当软件

3. 如果主控制器已经安装有Lab VIEW Real-Time及NI-RIO，选择**Custom software installation**并点击**Next**。当出现警告框时点击**Yes**。点击 **Ind Com for Ether CAT 扫描引擎 Support**旁边的框。自动检查需求的依靠项。点击**Next**在控制器上继续安装软件。
4. 如果软件安装完成，在Configuration面板的Remote Systems下选择控制器。点击**Advanced Ethernet Settings**按钮。在Ethernet Devices窗口，选择辅助MAC地址（为注明主地址的一项）。在Ethernet Device Settings下，选择**Mode**下拉选框中的**Ether CAT**并点击**OK**。

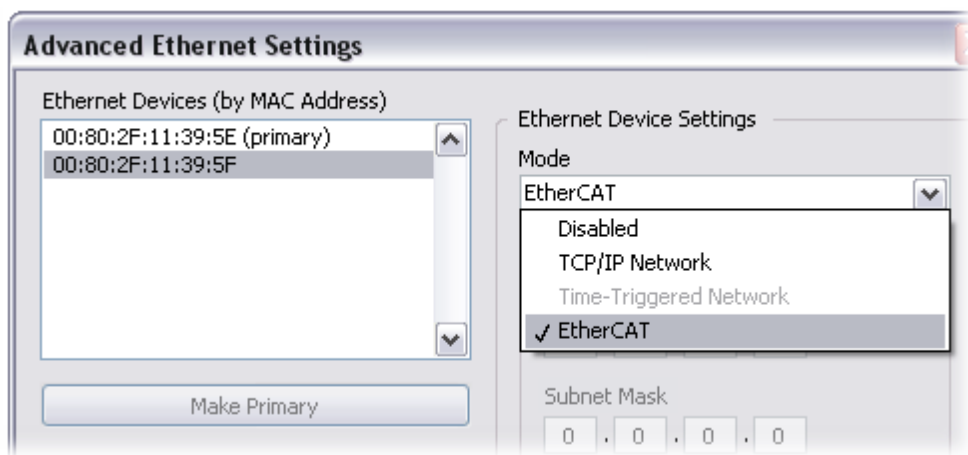


图5.11. 选择“Ether CAT”作为Compact RIO controller第二个以太网端口的模式

步骤2 添加确定性I/O至IO扫描

1. 将主机与Compact RIO控制器端口1连接至相同的以太网络。使用标准以太网线将Compact RIO控制器端口2连接至NI 9144扩展机箱的IN端口。为添加更多NI 9144机箱，使用以太网线将以上NI 9144的OUT端口连接至下一个NI 9144的IN端口。
2. 在Lab VIEW Project Explorer窗口，右击Compact RIO控制器并选择**New» Targets and Devices**。
3. 在Add Targets and Devices对话框，展开**Ether CAT Master Device**目录以自动发现主控制器上的Ether CAT端口。在弹出的Scan Slaves对话框，选择第一项来自动发现任何连接至控制器的从属设备。点击**OK**。Lab VIEW Project现在列出了主控制器，NI 9144机箱，I/O模块及每个模块上的物理I/O。

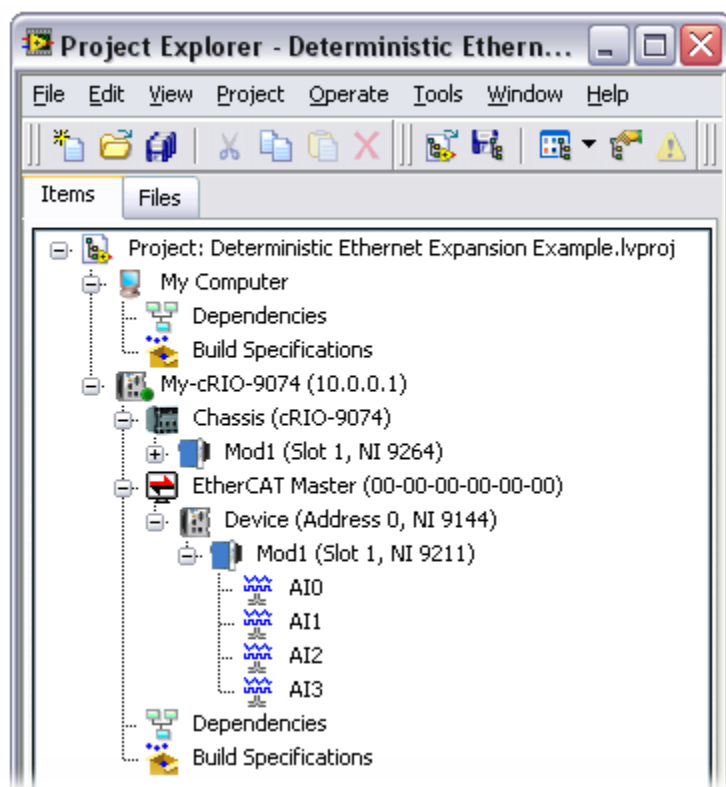


图5.12. Lab VIEW Project列出了主控制器，NI 9144机箱和I/O模块

4. 最后创建与物理通道相符的I/O别名，并使用这些别名来访问扩展IO。扫描引擎自动处理I/O同步，因此全部模块在每个扫描周期内同时被读取并刷新。

贴士：对于高通道数系统，你可以使用Multiple Variable Editor，通过输出和输入.csv的电子表格文件来合理化多I/O别名的创建。

步骤3. 添加FPGA Intelligence至确定性扩展机箱

1. 在Lab VIEW Project Explorer窗口，右击Device（NI 9144项目）并选择New» FPGA Target来创建你的NI 9144下新的FPGA Target。

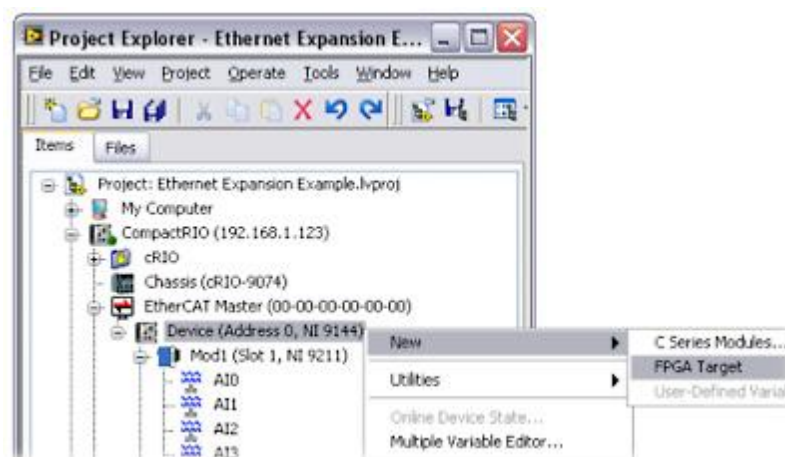


图5.13. 添加FPGA Target 至NI 9144机箱

2. 为使用Lab VIEW FPGA Mode中的模块，将模块从NI 9144拖放至FPGA Target。在Project Explorer窗口，你可以在Device和FPGA Target之间拖放模块以切换扫描模式和FPGA模式。

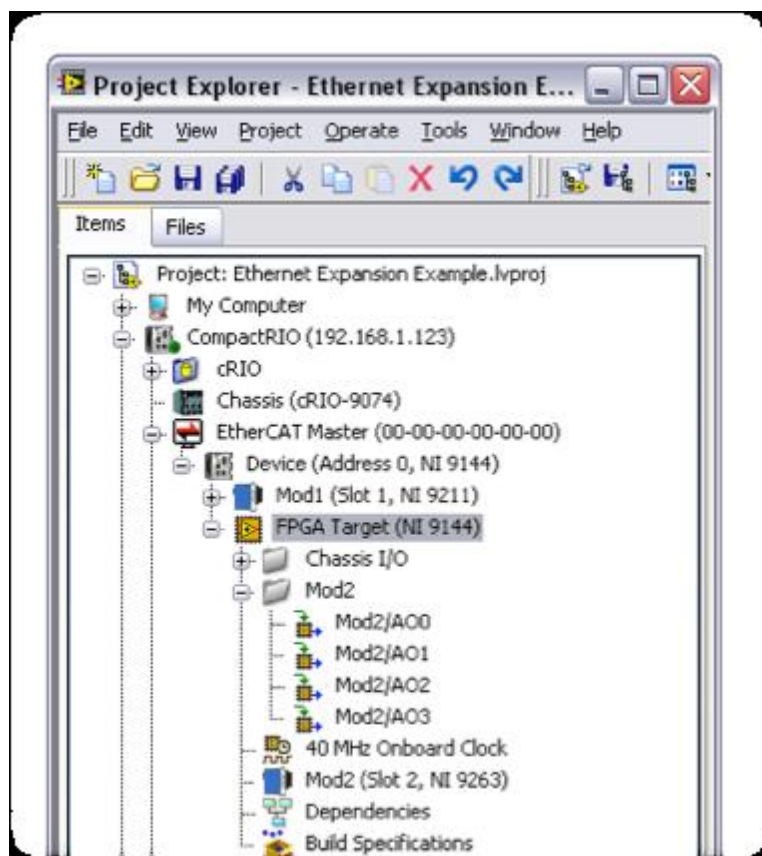


图5.14. 拖放模块至FPGA Target以在Lab VIEW FPGA Mode下编写它们

- 在Lab VIEW Project Explorer窗口，右击FPGA Target (NI 9144)并选择 **New» VI**来创建NI 9144上新的FPGA VI。你可以从FPGA Target (NI 9144) 下列出的模块中拖放FPGA I/O通道，至新的FPGA VI框图中。

Lab VIEW FPGA Programming API中的区别

如果你熟悉Lab VIEW FPGA中Compact RIO及其它NI RIO平台的编程，你需要了解局部FPGA编程及扩展I/O FPGA编程的一些区别。从Lab VIEW 2009开始，采用user-defined I/O variables同步FPGA数据及NI扫描引擎。这些用户自定义I/O变量也是在控制器实时VI与NI 9144 扩展机箱的FPGA VI间传递数据的唯一途径。关于更多用户自定义I/O变量的信息，参阅第6章。

FPGA 传递方法	局部机箱	扩展机箱
用户自定义I/O变量	✓	✓
FPGA 主机接口	✓	—
DMA 传递函数	✓	—
FPGA 前面板调试	✓	—

表5.1. 局部机箱与扩展机箱FPGA传递方法对比

- 一旦将FPGA目标添加至Lab VIEW Project中的NI 9144，你便可以通过右击NI 9144并选择**Add» User-Defined Variable**创建一个用户自定义I/O变量。变量名称、数据类型及数据方向（由主机向FPGA，反之亦然）可从Properties窗口中设定。
- 若运行FPGA VI，在Lab VIEW Project的cRIO-9074上右击，并选择Deploy All以配置NI 9144机箱。右击Compact RIO控制器并选择**Utilities»扫描引擎 Mode» Switch to Configuration**，切换扫描引擎至配置模式，并点击FPGA VI. Run按钮。这将启动编译进程并下载FPGA bitfile文件至Ether CAT。你可能需要更新机箱的固件以运行用户自定义FPGA代码。如果出现要求你更新固件的错误信息，按照NI 9144 User Guide中“更新你的固件”部分操作。你可以搜索ni.com找到user guide。
- 当编译和下载进程完成时，你须要将扫描引擎设回激活状态，以执行FPGA VI。按此操作，应右击你的Compact RIO控制器并选择**Utilities»扫描引擎 Mode» Switch to Active**。

请注意，能够在FPGA Mode中创建的用户自定义I/O变量是有数量限制的。NI 9144能为Scan Mode下的I/O变量和FPGA Mode下的用户自定义I/O变量保存总计512字节的输入数据和512字节的输出数据。例如，如果你在Scan Mode下使用4个32通道模块，同时每个通道占用32比特数据，则Scan Mode中 I/O变量使用256字节的输入数据。对于余下的256比特输入数据，你可以在FPGA Mode下创建64个个输入用户自定义I/O变量（同为32字节长）。

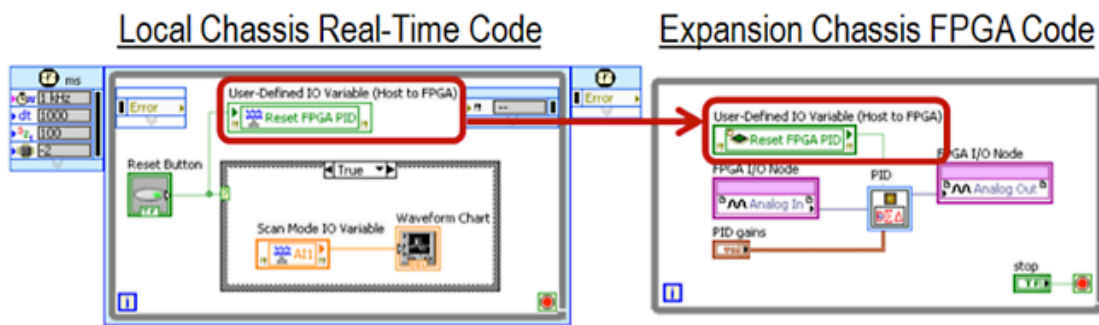


图5.15.创建用户自定义I/O变量，实现real-time与FPGA VI间的通讯

与使用扩展机箱实现户自定义I/O变量的要求不同，你可以用任意Lab VIEW FPGA算法函数对扩展机箱编程。此外，NI 9144机箱提供若干附加机箱I/O信号，帮助你自定义及同步代码。例如，你可以执行7个Ether CAT状态之一的指定的FPGA逻辑。因此，如果与主控制器的通讯停止，NI 9144机箱能够使用你的FPGA代码进入一个不同的操作状态。Ether CAT 驱动的NI-Industrial Communications中包含 Lab VIEW实例来说明高级FPGA特性，例如扫描引擎同步、专业数字信号和异步超采样。

机器视觉/检验

机器视觉是从一个或多个工业照相机获取影像并进行影像加工的综合过程。这些影像通常采用影像处理函数库进行处理，从简单的探测物体边缘到读取不同类型的文本或复杂代码。

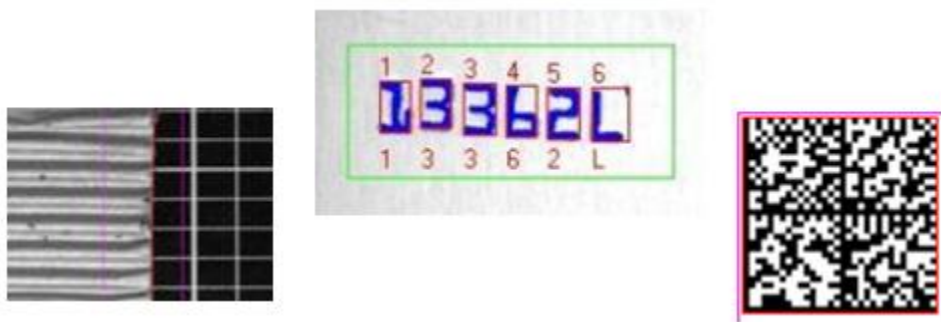


图5.13. 边缘探测，光学字符辨识和2D代码读取普通的机器视觉任务

通常，需要在一套视觉系统上由一个或多个影像进行多种测量。你可以将它用于多种应用，包括验证容器内的物品与瓶子上面的文字相符，并保证条码印刷在标贴正确的位置上。

来自处理影像的信息被装入控制系统，用于数据记录、损伤检测、动态向导、进程控制等。

关于NI视觉工具中可用的算法信息，参见[Vision Concepts Manual](#)。

机器视觉系统架构

典型的机器视觉系统包括一套连接到实时视觉系统的工业照相机，通常通过标准化照相机总线连接，例如IEEE 1394、千兆位以太网或Camera Link。实时系统进行处理影像，并具有可用于与控制系统通讯的I/O。

有些公司将照相机连接至视觉系统，创建智能照相机。智能照相机是具有有机载影像处理并包含一些典型的基本I/O的工业照相机。

National Instruments提供两种嵌入式机器视觉系统。NI Compact Vision System（图5.14）是一个实时嵌入式视觉系统，其特点是能

够与多达3个IEEE 1394照相机或29个用于同步和触发的通用I/O直接连通。该系统的特点是将以太网连接至你的工业网络，允许恢复NI Compact RIO硬件的通讯。

NI Smart Cameras（图5.14）是与可编程处理器连接的工业影像传感器，用以创建坚固的一体化机器视觉应用解决方案。这类照相机具有VGA（640x480像素）或SXGA（1280x1024像素）的分辨率，并有机会嵌入一套用于特殊算法附加性能的数字信号处理器（DSP）。这种照相机的特点是双千兆以太网端口、数字输入输出及内置照明控制器。



图5.14. NI Compact Vision System和 NI Smart Camera

National Instruments同样提供名为帧捕获器插件影像采集板，提供了工业照相机与PCI, PCI Express, PXI 及 PXI Express插槽的连接。这些采集板通常用于科学与自动化测试应用，但你也可以将其用于购买任何工业视觉系统和智能照相机前PC上的视觉应用原型的创建。

全部NI影像采集硬件采用相同的驱动软件，名为NI Vision Acquisition software。应用这一软件，你可以根据选择在硬件平台上设计并创建应用原型并配置到工业平台，使其最适合你的应用需求而无需大量更改代码。

照明和光学

本初级手册并未过多深入照明和光学部分，但它们对于你的应用成败至关重要。以下文件涵盖了大多数基础的和某些高级的机器视觉照明概念：

- 一个机器视觉照明的实用手册– 第一部分
- 一个机器视觉照明的实用手册– 第二部分
- 一个机器视觉照明的实用手册– 第三部分

机器视觉应用中的镜头改变了视场。视场是由照相机获取的影像形成的监视区域。确保你的视场系统包含你需要检查的目标是十分关键的。为计算影像系统的水平和垂直视场，使用方程1及照相机的影像传感器技术参数。

$$FOV = \frac{Pixel\ Pitch \times Active\ Pixels \times Working\ Distance}{Focal\ Length}$$

方程1，视场计算

式中，

- FOV是水平或垂直方向的视场，
- Pixel Pitch是水平或垂直方向相邻像素的中心距测量值，
- Active Pixels是水平或垂直方向的像素数量，
- Working Distance是镜头前部组件（外镜片）与监视目标的距离，
- Focal Length是镜头聚焦或分散光线的强度。

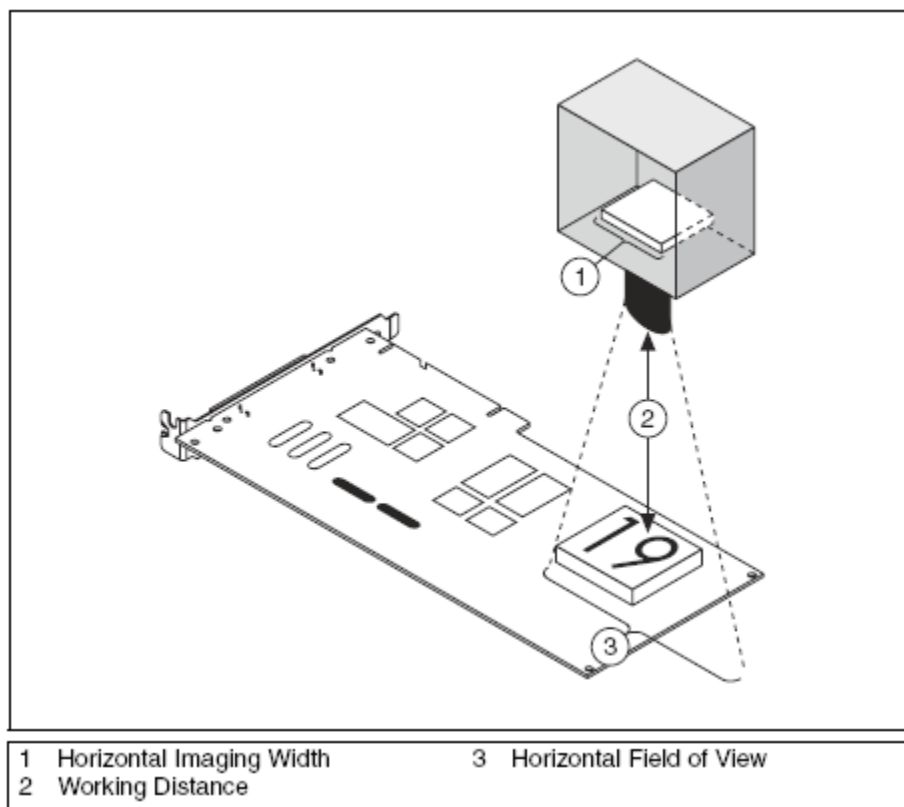


图5.15. 镜头选用决定了视场

例如，如果你影像设置的working distance为100mm，镜头的focal length为8mm，则使用全扫描模式下VGA传感器的NI Smart Camera水平方向的视场为：

$$FOV_{horizontal} = \frac{0.0074 \text{ mm} \times 640 \times 100 \text{ mm}}{8 \text{ mm}} = 59.2 \text{ mm}$$

同样，垂直方向的视场为：

$$FOV_{vertical} = \frac{0.0074 \text{ mm} \times 480 \times 100 \text{ mm}}{8 \text{ mm}} = 44.4 \text{ mm}$$

对于这个结果，你可能需要调整FOV方程中的多个参数，直到获得与你的监视要求相符的正确的部件组合。这可能须要增加你的working distance，选择更短focal length的镜头或更换高分辨率照相机。

软件选项

一旦为机器视觉设计选择硬件平台，就须要选择你想使用的软件平台。National Instruments为机器视觉提供两种应用开发环境（ADEs）。NI Compact Vision Systems和NI Smart Cameras均为NI Smart Cameras目标，所以您可以通过使用Lab VIEW Real-Time Module和NI Vision Development Module开发机器视觉应用。

NI Vision Development Module是一个机器视觉函数库，包括从基本滤波到模式匹配和光学字符辨识等功能。这个函数库也包含NI Vision Assistant及Vision Assistant Express VI。Vision Assistant是一个为机器视觉应用快速设计原型的工具。利用这一工具，您可以通过单击-拖动配置菜单来设定你的大多数应用。利用Vision Assistant Express VI，您可以在Lab VIEW Real-Time内使用相同的原型设计工具。

另一个软件平台选项是Automated Inspection (AI) 中的NI Vision Builder。Vision Builder是一个基于状态图模型的可配置机器视觉ADE，因此循环和决策制定都极其简便。Vision Builder AI的特点是很多高级工具建立在Vision Development Module内。两种硬件目

标均对Vision Builder AI有效，可使你灵活地选择最合适的软件以及最适于应用的硬件。

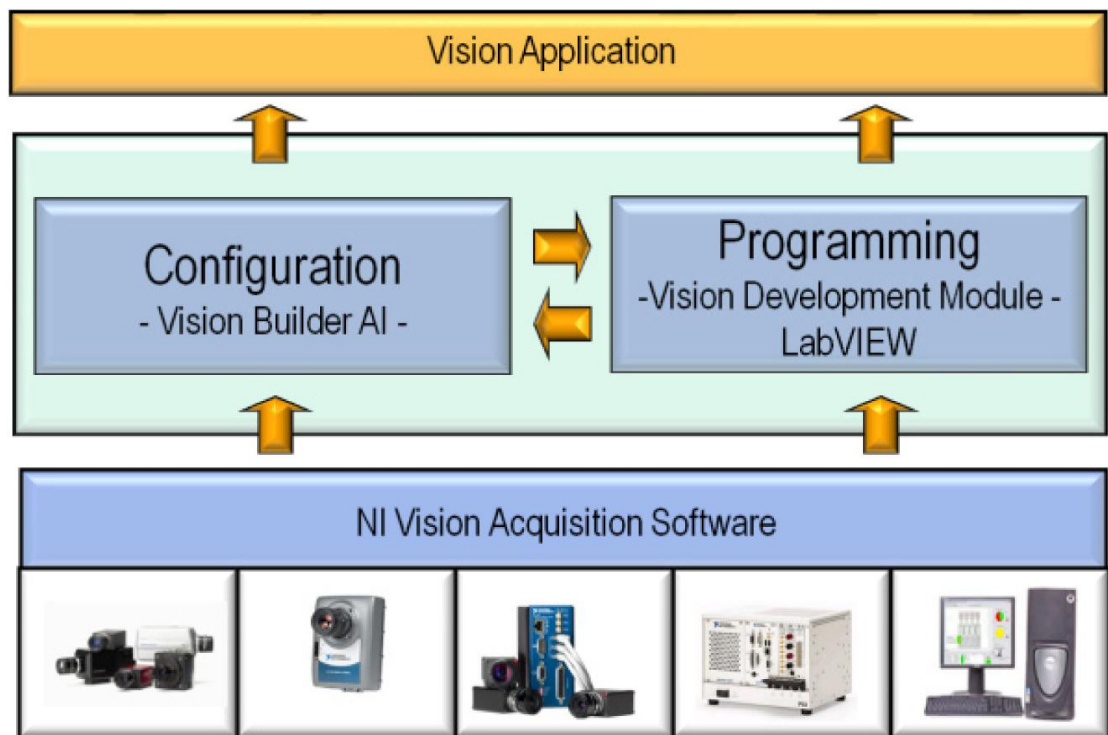


图5.16. National Instruments为机器视觉应用开发提供配置软件及完全编程环境

机器视觉/控制系统界面

大多数智能照相机或嵌入式视觉系统应用提供了实时内处理并给出输出结果，你可以将其用作控制系统的另一个输入。控制系统通常能够通过对视觉系统发送触发，控制影像采集和处理的开始时间。触发也可以来自硬件传感器，例如近距离传感器或正交编码器。

影像被处理为一组有用的结果，例如传送带上箱子的位置或印在汽车零件上2D编码的数值与质量。这些结果的报告返回控制系统，并且/或者发送至工业网络进行存储。你可以选择发送这些结果的方法，从一个简单的数字I/O到共享变量或者直接TCP/IP通讯，如该文档先前所述。

使用Lab VIEW Real-Time的机器视觉

以下例子说明为NI Smart Camera使用Lab VIEW Real-Time和Vision Development Module的机器视觉应用的开发。

步骤1. 添加一个NI Smart Camera至Lab VIEW Project

你可以像Compact RIO系统一样添加NI Smart Camera至Lab VIEW Project。如果你希望不连接智能照相机而设计原型，你可以模拟一个照相机。

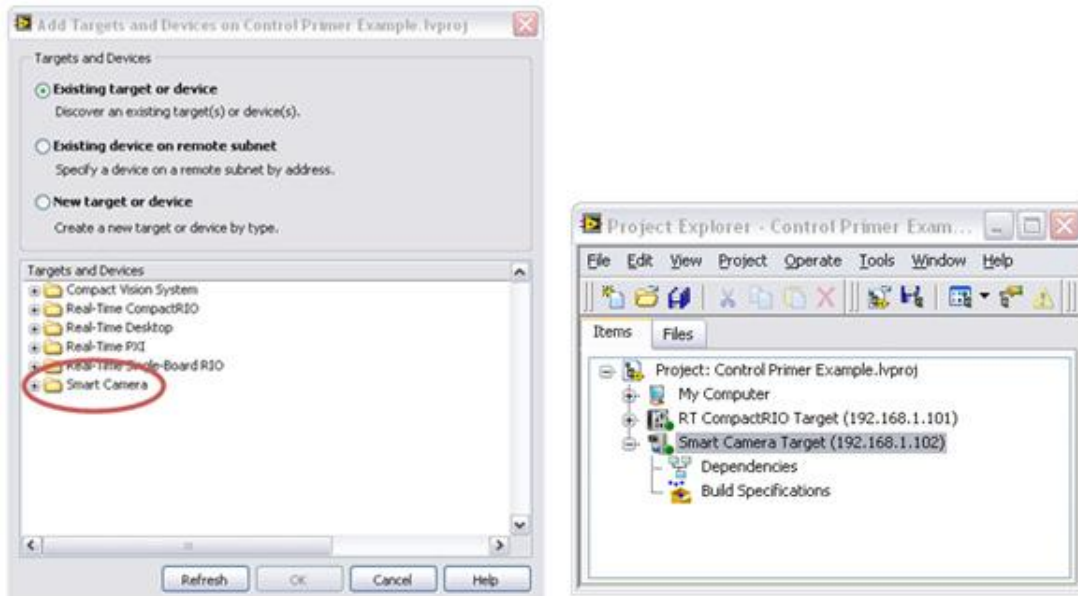


图5.17. 你可以像Compact RIO系统一样添加NI Smart Camera至Lab VIEW Project

步骤2. 使用Lab VIEW对NI Smart Camera编程

为智能照相机创建应用与为Compact RIO实时控制器创建应用几乎相同。主要的区别在于NI Vision Acquisition驱动的使用，来获得Vision Development Module 中的影像和算法以处理它们。

你可以创建新的VI并指向智能照相机，正如已经为Compact RIO创建的VI那样。

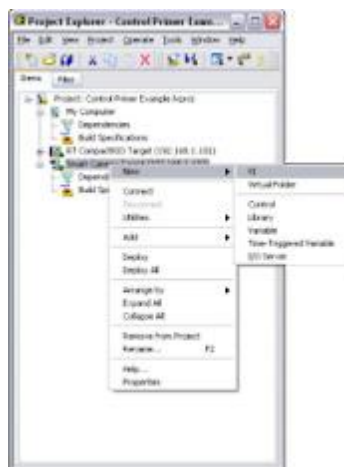


图5.18. 在Lab VIEW Real-Time中的NI Smart Camera to Your Project添加VI

根据部署，这一VI在运行时存入智能照相机的存储器并在智能照相机上运行。

为简化影像采集和处理的过程，National Instruments在Vision Palette中包括Express VI。在本例中使用这些Express VI从智能照相机（或视觉系统，如果你正在使用）采集影像并进行影响处理。访问这些Express VI，右击框图并选择 **Vision »Vision Express**



图5.19. Vision Express Palette

第一步是由照相机设置影像采集。如果你的照相机不可用或安装不正确，你也可以通过打开存储在硬盘上的影像来设置模拟采集。

将Vision Acquisition Express VI放在框图中。设计好的菜单驱动界面使你可以通过迅速获得首张影像。如果你有连接至电脑的可识别的影像采集硬件，则出现相关选项。如果没有，则在首菜单上出现“从磁盘打开影像”的选项。

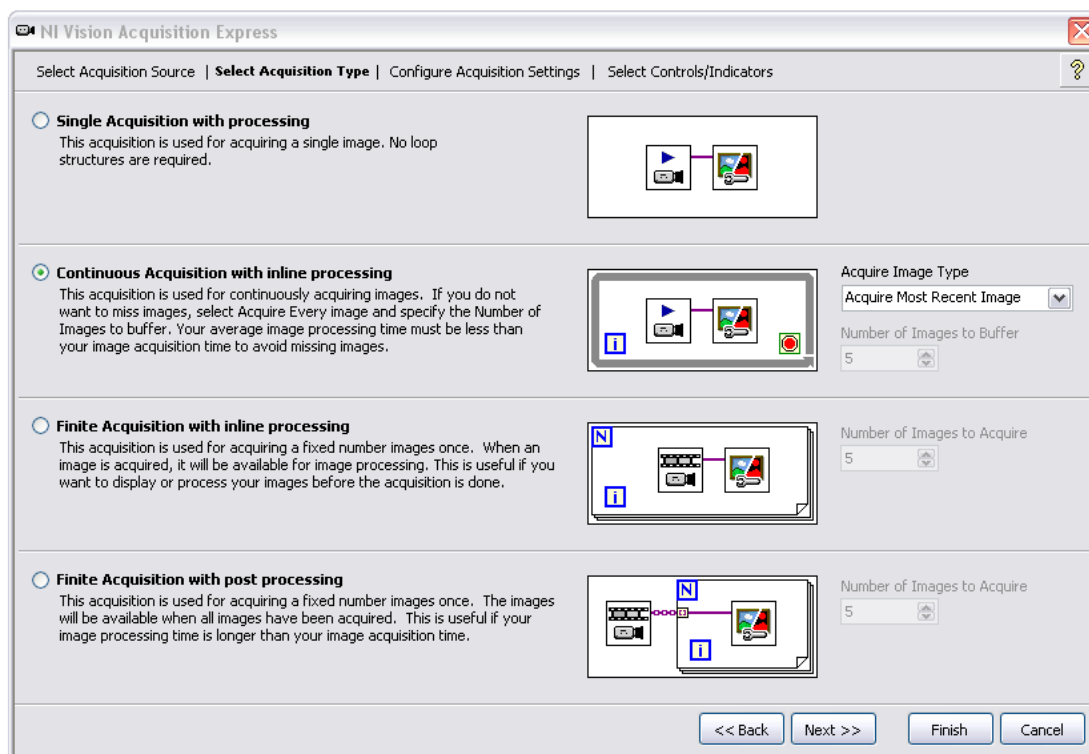


图5.20. Vision Acquisition Express VI指导你在Lab VIEW中创建视觉应用

下一步，选择你正在执行的采集类型。例如，选择Continuous Acquisition with inline processing。这允许你进入循环并处理影像。其次，测试你的输入来源并验证其正确性。至此，点击底部的完成按钮。

一旦这个Express VI在幕后产生Lab VIEW代码，框图会再次出现。现在，将Vision Assistant Express VI放在Vision Assistant Express VI的正右方。

根据Vision Assistant，可以迅速设立你的影像处理原型。你可以为实时系统配置相同的工具，虽然在实时系统中执行传统Lab VIEW VI具有更高的效率。概览Vision Assistant中的可用工具，参阅[NI Vision Assistant Tutorial](#)。

步骤3 与Compact RIO系统通讯

一旦设定好了可用Vision Assistant Express VI执行的机器视觉，最后要做的事情便是与Compact RIO的数据通讯。你可以使用网络发布的共享变量在两套系统间传递数据。本文档之前一部分深入介绍了Lab VIEW系统间的网络通讯。本例为检验电池夹，你需要获

得小孔状态（钻孔是否正确）以及当前电池夹内的间隙。

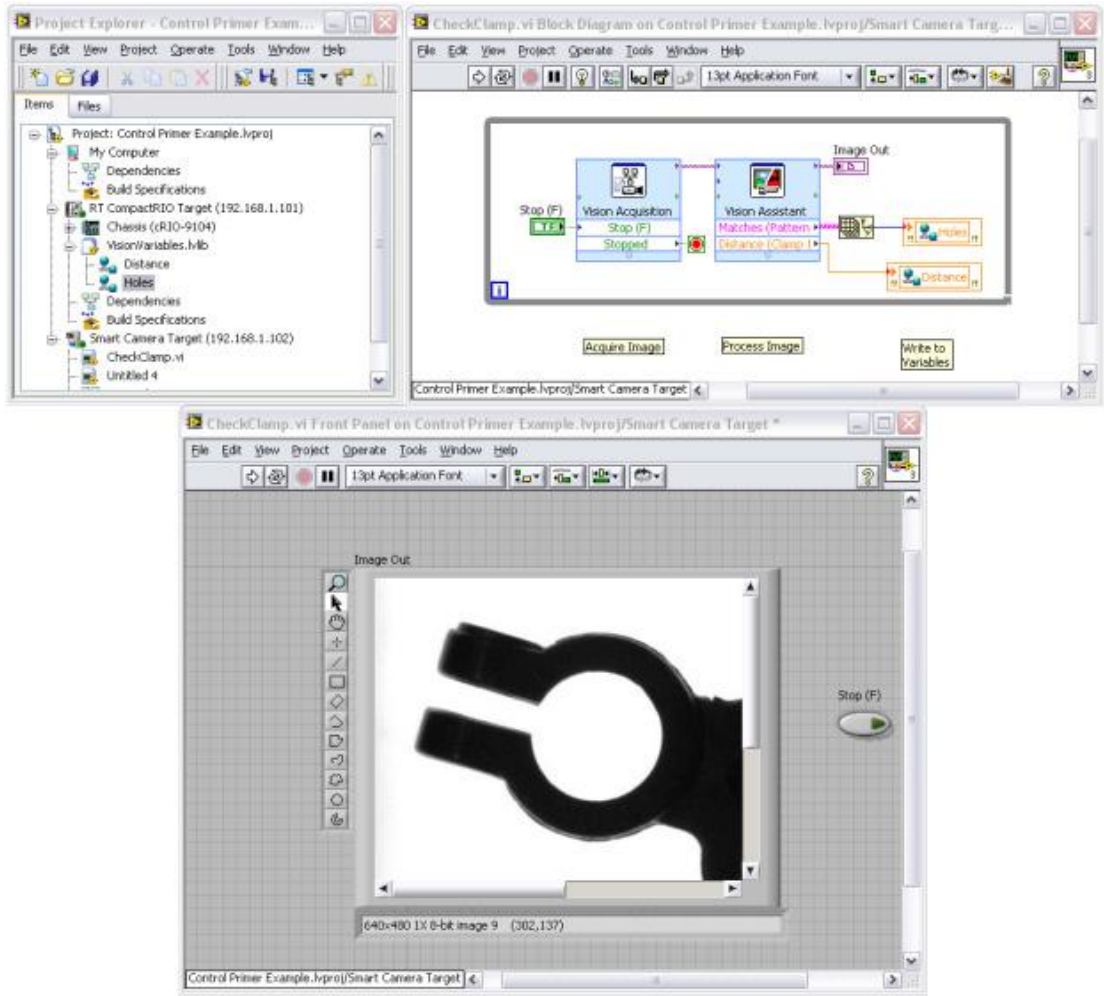


图5.21. 一个Lab VIEW的完整检验程序

如图5.21所示，检验结果作为当前值通过寄存在Compact RIO上的共享变量传递给Compact RIO。你也可以将数据作为命令传递给Compact RIO。

使用Vision Builder AI的机器视觉

如上所述，Vision Builder AI是一个机器视觉的可配置环境。可通过配置菜单执行全部影像采集、影像处理及数据处理。

步骤1. 使用Vision Builder AI配置NI Smart Camera

当你打开操作环境时，可以看见启动界面，在这里选择执行目标。如果没有已连接的智能照相机，你可以模拟一个智能照相机。本例中选择以下选项。

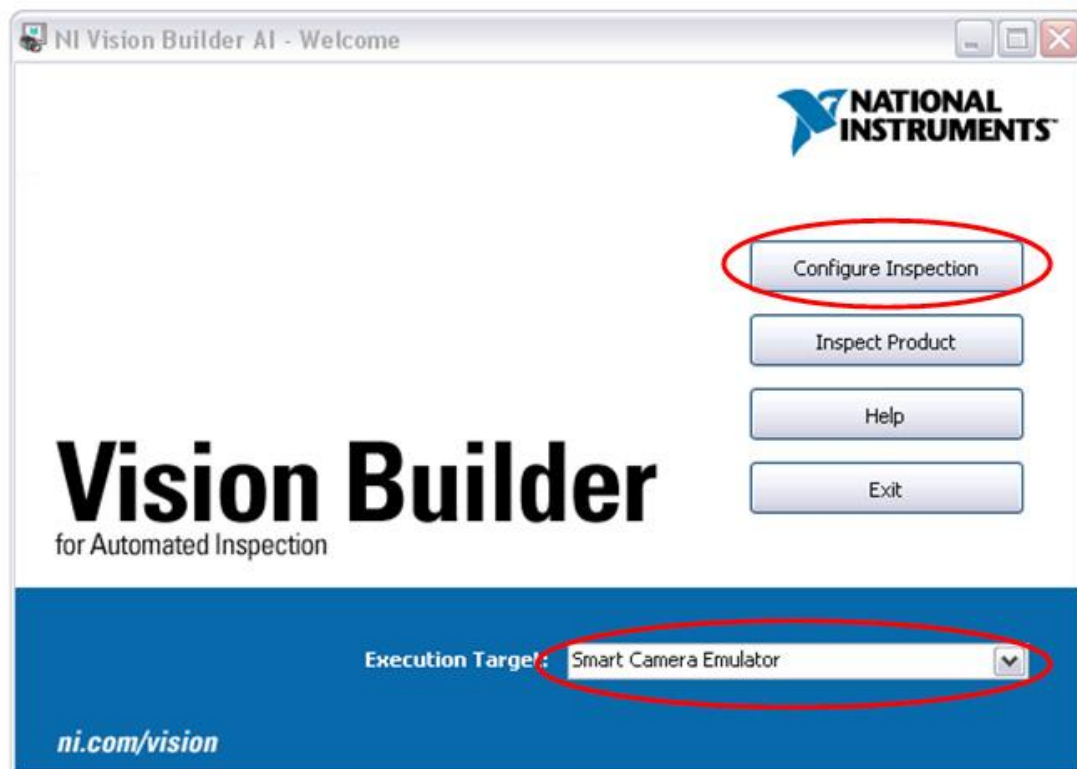


图5.22.由Vision Builder AI启动界面选择你的执行目标

使用这种仿真器，你可以设定照明和触发选项，安排与Compact RIO硬件通讯的I/O，同时完成很多配置智能照相机所需的其它动作，而无须对你的开发系统可用。在你已选择执行目标后，点击Configure Inspection。这将带你进入带有四个主窗口的开发环境。使用最大的窗口Image Display Window，查看你已采集的影像以及该影像上的覆盖图。使用位于右上角的窗口State Diagram Window，显示你的检查状态（当前状态高亮显示）。右下角的Inspection Step Palette窗口显示你的应用的全部检查步骤。最后，底部的窄条是Step Display Window窗口，可以看到当前状态的全部步骤。

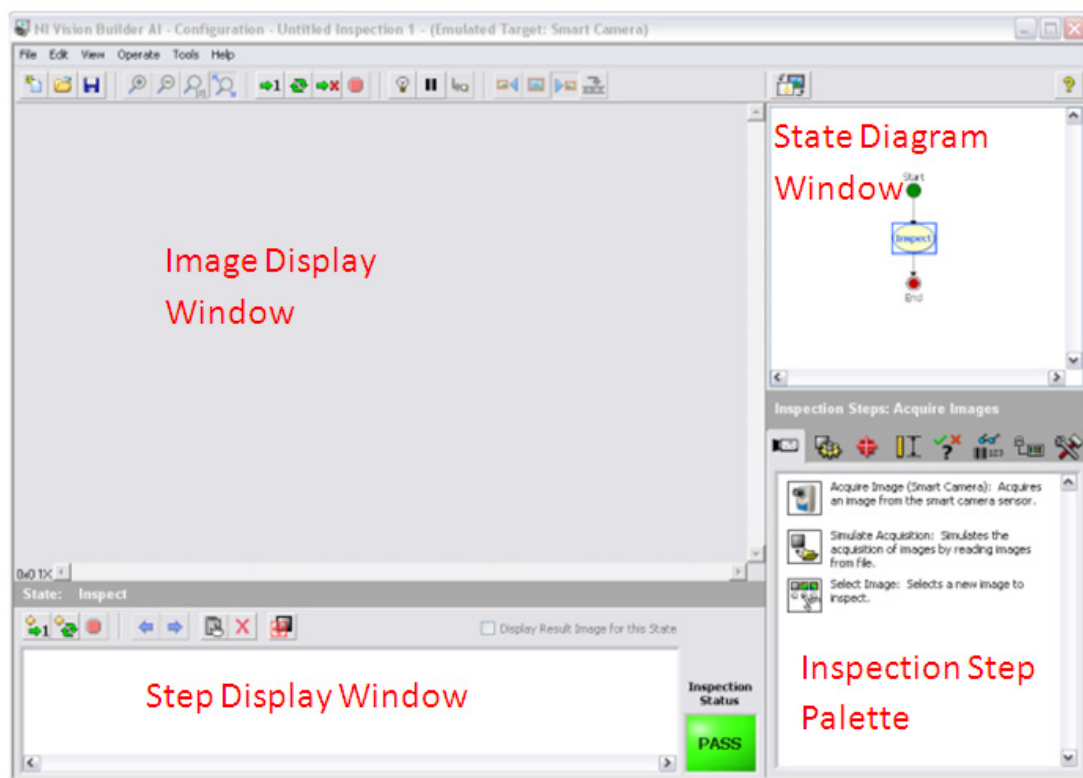


图5.23. Vision Builder AI Development Environment

本例与Lab VIEW实例执行同样的检查，检查了电池夹并将小孔数量及电池夹内的间隙大小通过两个寄存于Compact RIO硬件的共享

变量返回给Compact RIO。

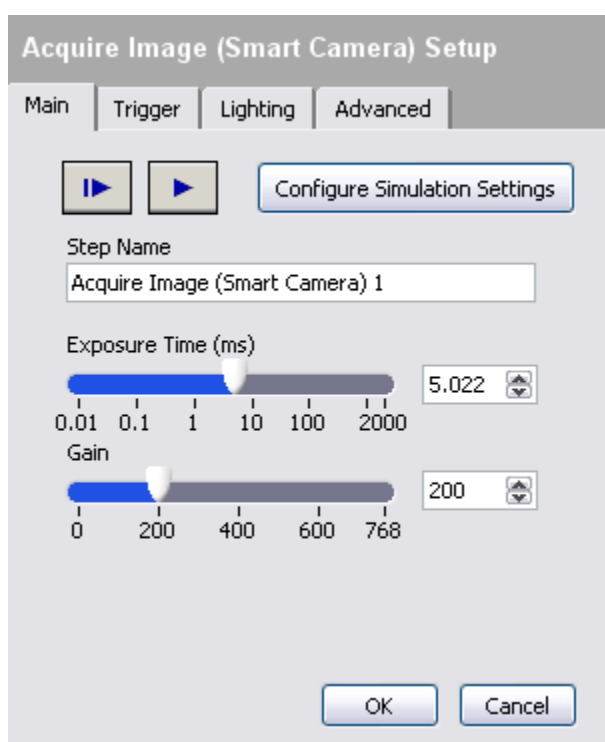
步骤2 配置检测任务

正如Lab VIEW中，检查的第一步是采集影像。利用Acquire Image (Smart Camera) 来执行这一步。通过点击**Configure Simulation Settings**按钮来选择此步并配置仿真器。例如，设置影像来获取以下路径的影像：

C:\Program Files\National Instruments\Vision Builder AI 3.6\Demolmg\Battery\BAT0000.PNG

这个影像库应该安装每一个Vision Builder AI（不同版本号）副本。选择上面的文件后，还要确定检查**Cycle through folder images**箱。

由此窗口还能发现，你可以配置曝光时间、增益以及其它相机设置。这些设置不会对仿真器产生任何影响，对在现实中，它们与照明和光学选项是相辅相成的。



5.24. 影像采集设置步骤

在这里设置匹配模式、边缘检测、对象检测、代码读取或你需要的任意其它算法。例如，执行模式匹配来设定物体的整体旋转，执行对象检测来查看小孔是否位于电池夹上，并利用卡尺工具来检测电池夹头之间的距离。这会产生一些显示是否通过的结果，你可以用来设置全部检查状态，并在覆盖层上显示给用户。

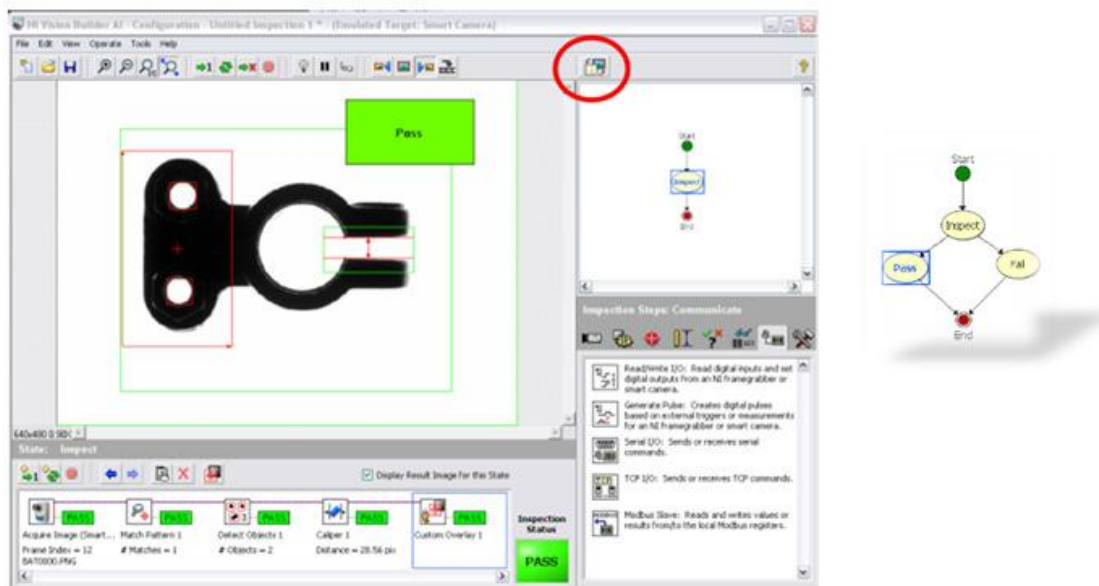


图5.25. Vision Builder AI中完成的检查

现在通过点击触发主窗口视图按钮（红色环形按钮）创建两个新的状态。创建一个通过状态与一个不通过状态。在这两种状态中，将数值发送报告回Compact RIO，但在不通过状态下，拒绝使用某些数字I/O的部分。

步骤3 与CompactRIO系统通讯

如之前讨论，Vision Builder AI提供多种类型I/O的接口，包括网络发布的共享变量，RS232，Modbus，Modbus/TCP，及原始 TCP/IP。本例中，使用变量管理器来接入寄存于CompactRIO上的共享变量。通过**Tools»Variable Manager**进入变量管理器。

这里你可以通过**Network Variables**表发现CompactRIO上的变量。在那里选择并添加变量并绑定至检查内的变量。

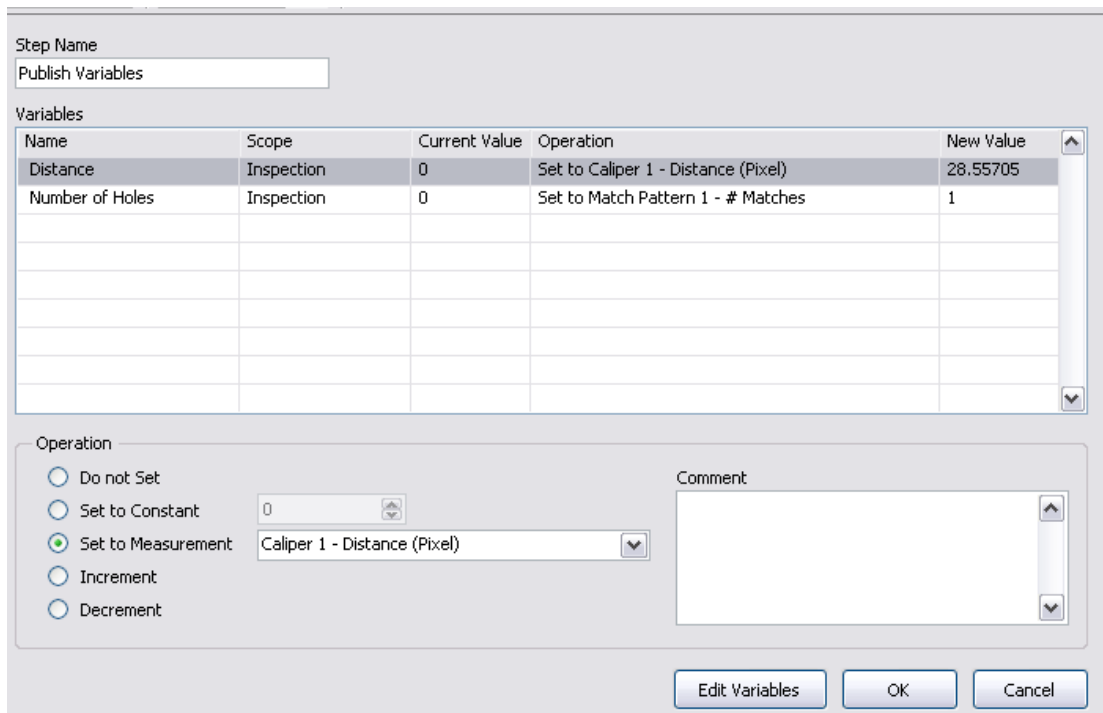


图5.26.在Vision Builder AI中设定变量通讯

现在，这些结果被送回CompactRIO，而检查将等待下一个照相机触发。

如你所见，两种方法（可编程及可配置）都为使用者提供了影像采集、处理的方法，然后使用相关信息将结果报告至CompactRIO控制系统或直接由实时视觉系统控制I/O。

运动控制

本文研究精确运动控制。马达控制是一种开关控制或简单的旋转设备的速度控制，如风扇和泵。你可以利用标准数字输出模块将马达控制设定为合适的马达启动器，或使用本文之前讨论的架构，利用虚拟输出将其设定为VFD。利用专用传感器、驱动器及快速控制循环，你可以通常在多轴系统上执行精确定位或快速运动控制。本部分讨论在NI CompactRIO硬件上执行高精度运动控制的复杂任务。

完整的运动控制实施过程是一个具有若干嵌套控制循环以及精确机械构件的复杂系统，其中一些循环以很高的速度运行。一个可靠的高性能运动控制系统包括以下设备：

1. 运动控制器：控制器控制运行软件算法的处理元件以及封闭的控制循环，用以产生基于运动约束的运动属性命令，该约束来自用户定义的应用软件及I/O反馈。
2. 通讯模块：此I/O模块与驱动和反馈设备连接。它将指令信号由运动控制器转化为驱动能够识别的数字或虚拟值。
3. 驱动/放大器：驱动/放大器由动力电子设备组成，可将来自通讯模块的虚拟或数字指令信号转换为用于旋转马达的电能。在很多情况下，它也带有一个关闭高速现行控制循环的处理元件。
4. 马达：马达将来自驱动/放大器的电能转换为机械能。马达扭矩常数 k_t ，以及马达效率决定了马达电流和机械扭矩输出间的比例。
5. 机械传动：机械传动由连接至马达的部件组成，这些部件引导马达旋转运动而进行工作。机械传动通常包括变速箱或滑轮以及导螺杆等机械装置，导螺杆可以将电机轴上的旋转运动根据一定的变速齿轮传动比转化为负载上的线性运动。皮带、传送带和阶梯为常用例子。
6. 反馈设备：编码器和限位开关等传感器为驱动/放大器及运动控制器提供瞬时的位置和速度信息。

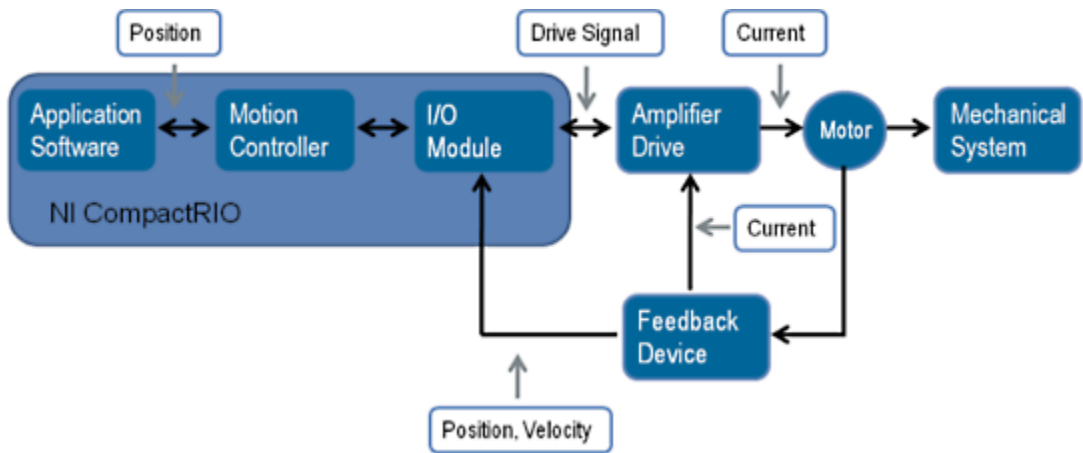


图5.27. 简化的CompactRIO运动系统示意图

运动控制器

运动控制器是运动系统的核心，包括运动控制软件，为你提供创建复杂多轴运动控制应用的灵活性。运动控制器由三个串联控制循环组成。

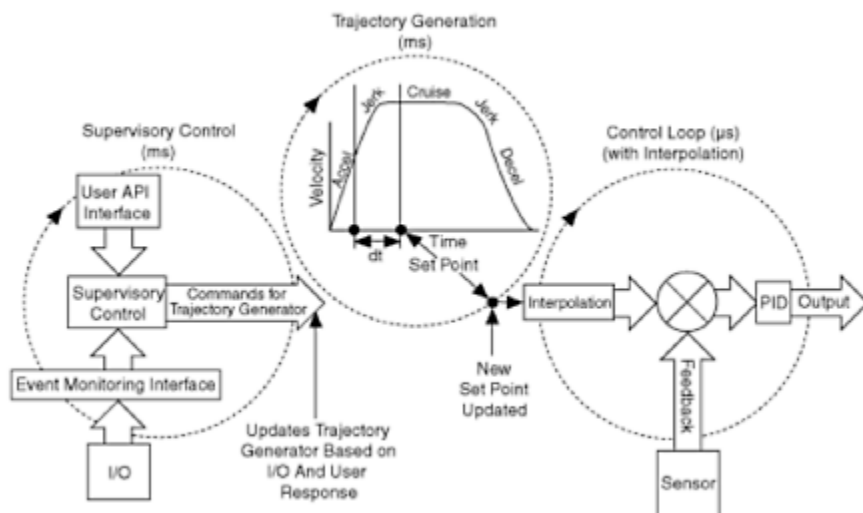


图5.28. NI运动控制器的函数架构

1. **Supervisory control:** 顶部控制循环执行命令排序并传递命令至轨迹生成循环。此循环执行以下步骤：
 - 系统初始化，包括返回起始位置
 - 事件处理，包括基于位置或传感器反馈和升级的触发输出
 - 故障检测，包括停止限位开关的动作，对紧急停止或驱动故障反应的安全系统，或其它监视动作
2. **Trajectory generator:** 此循环接收来自Supervisory control循环的命令，并基于用户指定属性生成路径规划。它为控制循环在确定样式内提供新的位置设定值。一般来说，此循环应该在5ms或更快的循环速率内执行。
3. **Control loop:** 此快速控制循环每50 μs执行一次。它利用位置和速度传感器反馈以及Trajectory generator的设定值为驱动创建命令。因为此训话运行速度高于Trajectory generator，它也会基于称为样条插值的程序生成中间设定值。对于步进式系统，控制循环被步生成元件取代。

LabVIEW NI SoftMotion及NI 951x驱动接口模块

你可以应用LabVIEW Real-Time及LabVIEW FPGA 模块从零开始建立全部运动组件，但National Instruments提供了一个软件模块以及C series I/O模块，其特点是具有预制和已检测组件。LabVIEW NI SoftMotion Module为运动控制编程提供了高级API，并提供如上所述的潜在运动架构作为驱动器服务。Supervisory control循环和trajectory generator在CompactRIO控制器的实时处理器上运行。NI还提供C Series驱动接口模块(NI 951x)，以运行控制循环并提供与驱动和运动传感器连接的I/O。全部循环和I/O模块由LabVIEW Project配置。

对于需要更多客户化的应用，例如自定义trajectory generator或不同的控制算法，LabVIEW NI SoftMotion提供打开代码的软件工具，并使用LabVIEW Real-Time和LabVIEW FPGA自定义这些组件。对于NI 951x C Series无法使用的特性和功能的应用要求，LabVIEW NI SoftMotion提供轴向接口节点，所以你可以使用其它I/O或第三方驱动的通讯接口。此外，LabVIEW NI SoftMotion允许运动应用的可视化原型设计以及和SolidWorks Premium 3D CAD设计应用连接的机器设计。对于SolidWorks，使用NI SoftMotion可以在实物原型设计产生费用之前，使用LabVIEW NI SoftMotion开发的实际运动属性来模拟SolidWorks中创建的设计。

了解更多信息，请访问ni.com/virtualprototyping。

CompactRIO上的运动入门指南

你可以按4个基本步骤使用CompactRIO建立一个运动控制应用。

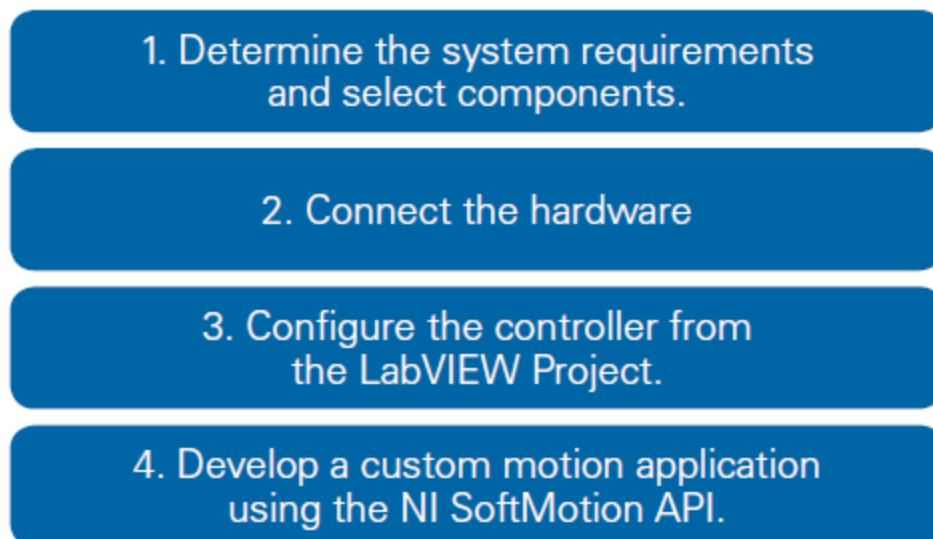


图5.29. 建立运动控制系统的四个步骤

确定系统要求并选择组件

开始时为系统选择正确的机械构件和马达。最普通的应用包括将目标从一个位置移动至另一个位置。将旋转马达的旋转运动转化为有用的线性运动的典型方法是将马达连至某工作台(stage)，并运用导螺杆机制来移动有效荷载。这些机械工作台(stage)能够提供用于定位和移动物体的线性或旋转运动。它们有多种类型和尺寸，因此可用于各种不同应用。为了找到适于应用的正确工作台(stage)，你必须熟悉一些描述这些工作台(stage)的常用术语。选择工作台(stage)时考虑的关键事项包括以下几方面：

- 传动齿轮比：决定了马达每个转动循环该工作台(stage)的线性行进距离。
- 精度：指令移动距离与标准距离的接近程度。
- 分辨率：系统能够执行的最小行程-可小至纳米级。
- 行程：系统在某方向能够移动的最大长度。
- 重复性：相同情况下某指令位置的重复运动。通常，单向重复性特指从一个方向回到相同点的能力，双向重复性特指从各自方向回到相同点的能力。
- 最大负载：该工作台(stage)物理上设计的能够承担的最大重量，考虑精度和重复性。

工作台(stage)选择

你可以为你的应用需求从多种工作台(stage)中选择。可以将工作台(stage)范围缩小至两种主要类型-线性和旋转。线性工作台(stage)沿直线运动并经常相互叠放以提供多方向运动。具有x, y, z 分量的多轴系统是一种用于在3D空间定位目标的通用设置。旋转工作台(stage)则绕轴（通常是中心）旋转。线性工作台(stage)用于物体的空间定位，而旋转工作台(stage)则用于物体在空间中确定方向并调整物体的转动、倾斜及偏转。很多诸如高精度准直的应用，需要定位与定向两种工作台(stage)来实现高精度瞄准。转动工作台(stage)的分辨率常以角度或弧分计算（1度等于60弧分）。也有测角仪之类的特殊类型工作台(stage)，它看起来像线性工作台(stage)，却沿弧度而非直线运动，又如六足机器，一个在六个方向运动的平行机制，可以控制x, y, z,转动，倾斜及偏转运动。利用六足机器，你可以定义一个可供工作台(stage)旋转的虚拟点。虽然六足机器具备这一优势，但其缺点在于它的平行机制，且相比于简单重叠工作台(stage)，它涉及到更加复杂的运动力学。

间隙

为精确运动系统选择工作台(stage)时另一个需要考虑的工作台(stage)属性是间隙。间隙是当系统中的一个齿轮改变方向，并在与相邻齿轮接触前移动一小段距离时产生的延迟。间隙能够导致明显的不精确，尤其在使用很多齿轮的传动系系统中。当移动纳米量级时，少量的间隙也能引起系统较大的不精确。有效的机械设计将间隙最小化，但也未必能将其彻底消除。你可以通过在软件中执行双回路反馈来抵消这一问题。NI 9516 C Series伺服驱动接口模块支持双编码反馈并针对单轴接收两个不同来源的反馈。为了解需使用这一特性的环境，设想一个工作台(stage)。如果你直接监控工作台(stage)位置（与马达驱动工作台(stage)的位置相反），便能看出你指令的运动是否达到目标位置。然而，由于运动控制器为马达而不是工作台(stage)提供输入信号，作为反馈的主要来源，预期输出相对于输入的差异可能造成系统的不稳定。为使系统维持稳定而进行必要微调，你可以直接从马达上的编码器监测反馈作为附属反馈来源。使用这种方法，你可以监测工作台(stage)的实际位置并解释传动系中的不精确。

马达选择

马达提供工作台(stage)的运动。一些高精度工作台(stage)和机械部件具有将间隙最小化并提高重复性的内置马达，但大多数工作台(stage)和部件使用机械耦合用以连接至标准回转马达。为使连接更加简便，National Electrical Manufacturers Association (NEMA)将马达尺寸标准化。对于分数马力电动机，框架尺寸使用两位数命名，例如NEMA 17 or NEMA 23。对于这些马达，框架尺寸表示特定的轴高度、轴直径和安装孔样式。框架命名不根据转矩和速度，所以对于同样的框架尺寸，可选择不同的转矩与速度的组合。

正确的马达必须与机械系统相匹配，以提供所需的性能。你可以从以下四种主要马达技术中选择：

- 1. **步进马达：**步进马达比相同尺寸的伺服马达更便宜且更易于使用。这些设备以离散步运动，因此成为步进马达。控制步进马达需要一个步进驱动，从控制器接收步进和方位信号。步进马达对低成本应用有效，可用于运行开环结构（无编码反馈）。一般而言，步进马达在低速下具有高扭矩，而在高速下具有低扭矩及良好的保持扭矩，以及较低的最大速度。低速的运动也可能是波动的，但多数步进马达具有微步进能力来最小化这一问题。
- 2. **有刷伺服马达：**这是一种简单马达，电触头通过称为整流器的机械旋转开关将能量传递至至电枢。这些马达提供双线连接，通过改变进入马达的电流来控制，通常为脉冲宽度调制（PWM）控制。马达驱动将通常为±10 V的模拟指令信号的控制信号，转化为进入马达的电流输出且可能需要调谐。这些马达十分易于控制且提供作用范围内良好的扭矩。然而，它们需要周期的电刷维护，且相比于无刷伺服马达，由于电刷的机械限制，它们的速度范围有限且效率较低。
- 3. **无刷伺服马达：**无刷伺服马达采用永久磁铁转子，三相驱动线圈及确定转子位置的霍尔效应传感器。专用驱动将来自控制器±10 V的模拟信号转换至马达的三相电源。驱动具有执行电子整流的智能，并且需要调谐。这些高效马达传递高扭矩和速度，而只需较少的维护。然而，它们的设置和调谐较为复杂，且马达和驱动更加昂贵。
- 4. **压电马达：**压电马达使用压电材料生成超声波振动或阶跃，并通过履带式运动产生线性或旋转运动。这些马达能生成非常精确的运动，经常用于诸如激光第等纳米定位应用。由于精度很高，压电马达常被融入工作台(stage)或驱动器中，你也同样可以使用压电转动马达。

工作台 (stage)驱动 技术	速度	最大负载	行程	重复性	相对复杂性	相对成本
步进	中/低	中/低	高	中/低	低	低

有刷伺服	高	高	高	中	中	低
无刷伺服	很高	高	高	高	高/中	高
压电	中	低	低	很高	高	高

图5.30. 马达技术选项对比

由于马达和驱动的连接十分紧密，你应该使用同一供应商的马达和驱动的组合。虽然这不是必要条件，但会使调谐和选择更加简便。

连接硬件

一旦选择了合适的机械、马达及应用的驱动，你需要将CompactRIO系统连接至硬件。利用Axis Interface Node 及 LabVIEW FPGA，你可以使用任意C Series模块连接至CompactRIO系统；然而，为简单起见，National Instruments推荐使用NI 951x C Series驱动接口模块。

NI 951x Drive Interface 模块

NI 951x模块为单轴提供伺服或步进驱动接口信号，并可连接至许多步进或伺服驱动。除了为运动控制提供合适的I/O外，模块还具有使用专利NI步进生成算法或伺服控制循环来运行样条插值引擎的处理器。

- NI 9512是使用增量式编码器反馈的单轴步进或定位命令驱动接口模块。
- NI 9514是使用增量式编码器反馈的单轴伺服驱动接口模块。
- NI 9516是使用双编码器反馈的单轴伺服驱动。

扫描模式下运行时，这些模块必须用在CompactRIO机箱的前四个插槽。而对于LabVIEW FPGA，你可以使用任意插槽。

NI 951x模块被设计用于简化连线，提供了将全部运动控制的I/O连接至单一模块的灵活性。为使连接更加便利，数字I/O可进行软件配置，以连接至沈流或源流工作台(stage)。模块提供以下功能：

- 模拟或数字命令信号至驱动
- 可驱动信号；沈流或源流（24V）的软件可编程
- DI信号DI signals for home, limits? ? ? ? 及一般数字I/O；沈流或源流（24V）的全部软件可编程
- 编码器输入和5V供电；可为单端或多端配置
- 用于高速位置获取/对比功能（5V）的DI/DO
- 为编码器状态，限制状态和轴故障提供快速调试的LED

为更加简化连线，National Instruments为连接NI 951x驱动接口模块至外部步进驱动或伺服放大器，提供以下几种选项：

- NI 9512-to-P7000 Stepper Drives Connectivity Bundle-连接NI 9512至来自NI的P70530 或 P70360步进驱动。
- NI 951x Cable and Terminal Block Bundle-将NI 951x模块与37插针的弹簧或螺栓接线端子连接。
- D-Sub and MDR solder cup connectors-简化自定义线缆创建。
- D-Sub to pigtails cable and MDR to pigtails cable-简化自定义线缆创建。



图5.31. 选择若干选项以简化NI 951x模块与驱动的直接连接

由LabVIEW Project配置控制器

为使用LabVIEW中的LabVIEW NI SoftMotion，你必须首先在LabVIEW Project中建立轴、坐标及表格。创建这些项目后，你将一个逻辑通道与物理运动来源相关联，并实例化后台控制循环以在控制器上运行。当使用NI SoftMotion LabVIEW API开发动作应用时，你可以使用你创建的逻辑通道。

- 轴是一个与单独马达相关联的逻辑通道。每个你控制的马达必须在轴上。
- 坐标系是一条或多条轴的组合。通过制定坐标系中的多轴方案，你可以创建多轴协调运动。例如，如果你有XY向工作台(stage)并希望创建椭圆，单独控制两个马达是困难的。但如果将它们组合成一个轴，LabVIEW NI SoftMotion轨迹发生器将自动为每条轴生成控制点，以使运动成为椭圆。
- 表格用于指定更多复杂运动属性，例如轮廓线运动或凸轮。你可以由外部带分隔符的文本文件导入运动属性。

添加轴，坐标及表格至Project

你可以在LabVIEW Project 中通过右击CompactRIO控制器并选择New来创建轴，坐标及表格。

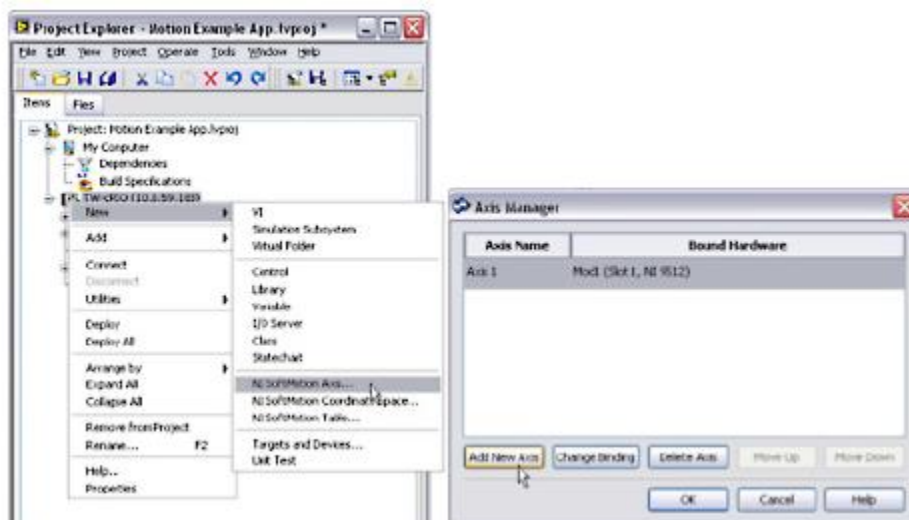


图5.32.在CompactRIO系统中添加轴

轴包括轨迹生成器，比例积分衍生（PID）控制循环或步进输出以及管理控制。你可以将LabVIEW NI SoftMotion轴与模拟硬件或真实硬件相关联。伺服轴线需要编码器反馈源。而开环步进轴不需要操作反馈。

创建轴之后，你可以生成坐标系并在坐标系中添加轴。当使用LabVIEW中的坐标源时，将以包含轴线信息的1D数组接收目标位置和其他坐标信息，其轴线信息按使用此对话框时轴线添加的顺序到达。

配置轴线

一旦将轴线添加至project，你需要通过右击来配置轴线并选择属性。

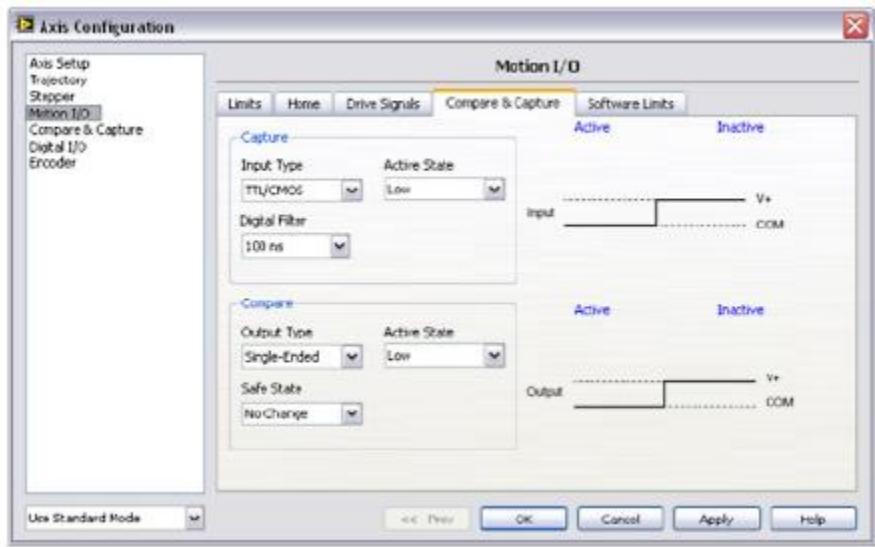


图5.23. 你可以使用轴线配置特性来配置全部运动I/O参数

为连接至NI P7000系列步进驱动的配置一个步进驱动，遵循以下几步：

- 1. 在Project Explorer窗口右击轴线并由快捷菜单选择Properties，打开Axis Configuration对话框。
- 2. 在Axis Setup页面，确定Loop Mode设定为Open-Loop。在开环模式下配置的轴线产生步进输出，而无需用来核实位置的马达反馈。
- 3. 同样在Axis Setup页面，确定Axis Enabled及Transition to Active Mode上的Enable Drive复选框带有选中标记。当NI扫描引擎切换至激活模式时，这些选项将轴线配置为自动激活。
- 4. 如果模块没有物理限制起始输入连接，你必须为正确的系统操作使这些输入信号失效。为使限制和起始输入无效，进入Motion I/O页面并移除Forward Limit, Reverse Limit及 Home部分中 Enable复选框中的选中标记。
- 5. 根据你的系统需求配置任意附加I/O设置。
- 6. 点击OK关闭Axis Configuration对话框。
- 7. 右击Project Explorer窗口中的控制器项目并由快捷菜单选择Deploy All以完成配置。

警告 确定生成全部硬件连接，并在配置project前打开电源。这些配置将NI 扫描引擎切换至激活模式并使轴线和驱动生效，如果连接，你便可以立刻启动一个动作。

测试运动系统

为确定动作配置及连接正确，你可以使用Interactive Test Panel来测试并调试你的运动系统。使用Interactive Test Panel，你可以执行简单的直线运动并监测运动和I/O状态信息，改变运动限制，获得系统中的错误和故障信息，并观察运动的位置和速度图。如果有连接至系统的反馈装置，你还可以获得反馈位置和位置错误信息。

为启动Interactive Window，右击Project Explorer窗口中的轴线并选择快捷菜单上得Interactive Test Panel。使用此选项卡设定需要的位置，运动模式和运动约束。

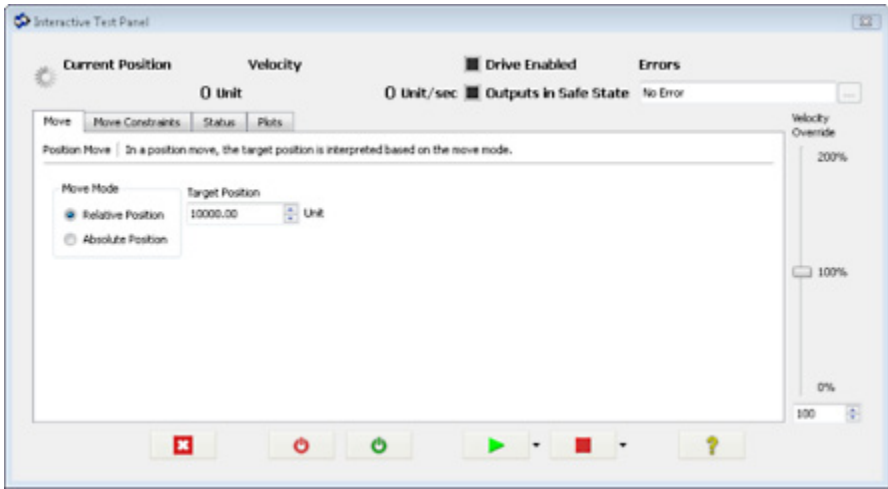


图5.34. 在Interactive Test Panel中，你可以在编写代码前验证运动配置

为执行运动，点击对话框底部的Start 按钮，开始配置好各项的运动。使用Status and Plots选项卡来监测进行中的运动。

使用LabVIEW NI SoftMotion API开发自定义运动应用

LabVIEW NI SoftMotion Module提供功能块API建立确定性运动控制应用。这些功能块被PLCopen运动功能块激励，使用与IEC 61131-3功能块相同的术语和执行范式。虽然功能块编程与LabVIEW数据流相似，在使用功能块创建应用前你也必须了解它们的一些执行区别。

功能块本身并不运行任何运动算法。相反，它们是一个API，发送命令至CompactRIO控制器上作为驱动服务运行的运动管理器。运动管理器以一定的扫描速率运行，但你可以任意你想要的速度运行功能块API，甚至以非确定性代码调用功能块。

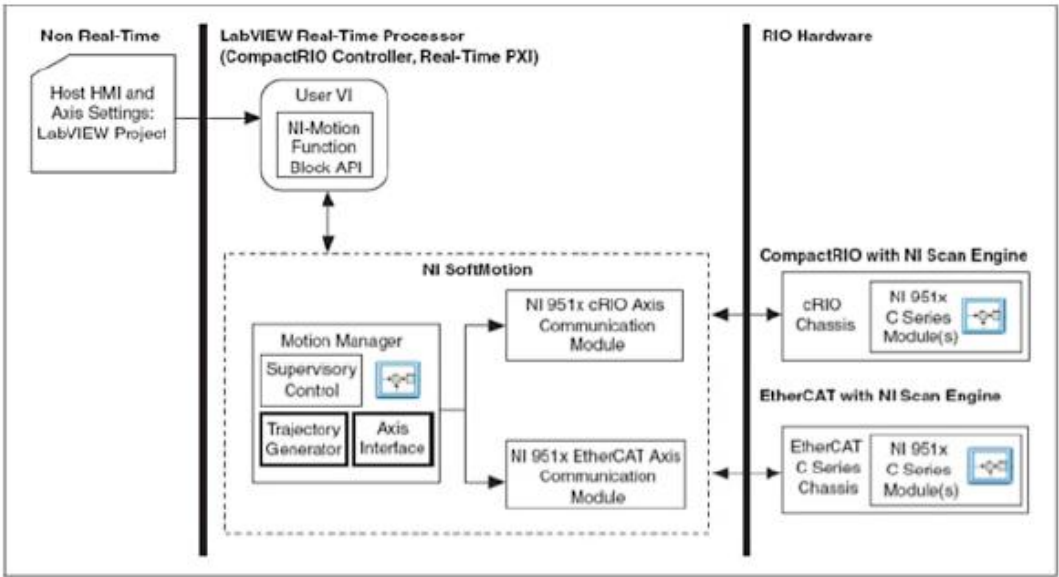
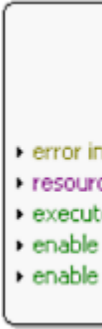


图5.35. CompactRIO 系统上的 LabVIEW NI SoftMotion Components框图



面板 对象	面板 符号	说明
----------	----------	----

图5.36. LabVIEW NI SoftMotion功能块

运动功能块是CompactRIO系统上运行的运动管理器的API。它们执行两个动作：

- 1. **发送命令至运动管理器。**当“执行”输入由低至高（上升沿）转换时，功能块发送指令至管理器。默认值为假，如果一个真常数连至程序块，首次迭代将作为上升沿。
- 2. **检查运动管理器。**每次迭代，功能块将检查管理器是否成功执行命令。检查结果通过“done,” “busy,” “active,” “aborted,” 及 “error out”等输出返回。

输出排斥

Busy, done, error out及aborted 是互相排斥的，且在任意给定时间设定。功能块的任意时刻只有一个输出可以为真值。如果execute输入为真，其中一个输出就必须为真。

输出状态

done 和aborted输出使用execute的下降沿重置 (当execute 为 “高” 时状态封锁). 然而， execute的下降沿接到命令后不会停止或影响运动管理器的执行。

Done输出的行为

当命令的动作成功完成后， done输出设为真值。默认值为假。一旦动作完成，当execute为假值时， done也被重置为假值。

Aborted输出的行为

当命令的操作被另一个命令中断时， aborted将被设置。 Aborted的重置行为与done相似。

Busy输出的行为

每个功能块都有指示功能块操作未完成的busy输出。 Busy设置在execute的上升沿，且在done, aborted,或 error out设定时重置。 建议此功能块执行的应用控制至少要在busy为真值时才可终止，因为软件可能被留在不确定状态。

输出激活

当功能块控制指定来源，全部属性被设定且功能块已被执行时， active 输出设定为真值。

Line		使用轴线或坐标系源执行直线运动。直线运动使用一条或多条轴线连接两点。运动特性根据 Straight-Line Move Mode 变化。
Arc		执行圆形，球形或螺旋弧运动。弧运动按照你所指定半径的圆弧产生运动。执行的弧运动根据 Arc Move Mode 变化。
Contour		使用轴线或坐标系源执行轮廓线运动。轮廓线运动以一系列位置表示，软件利用这些位置外推一条平滑曲线。这些位置存储在表格中。将运动的起始点视为临时“zero”位置，则运动中的每个点可看做一个绝对位置。轮廓线运动类型根据 Contour Mode 变化。
Reference		执行参考运动，例如在轴线源上定位起始或限制位置。参考运动用于初始化运动系统并建立可重复参考位置。参考运动特性根据 Reference Move Mode 变化。
Capture		记录基于外部输入的编码器位置，例如传感器状态。你可以利用捕捉的位置来执行与其相关的运动，或仅记录捕捉事件发生时编码器的位置。
Compare		使马达与外部活动及指定的编码器位置同步。当到达指定位置时，执行用户配置的脉冲。位置比较操作的特性根据 Compare Mode 变化。
Gearing		为传动操作配置指定的轴线。传动将附属轴线的运动与主设备的运动同步，可以是一个编码器或其它轴线的轨迹。附属轴线可能以比主设备更高或更低的传动比运动。例如，主轴线的每次转动可能引起附属轴线转动两次。执行的传动操作类型根据 Gearing Mode 变化。
Camming		为凸轮操作配置指定轴线。这些比例由 LabVIEW NI SoftMotion 自动处理，允许传动比的精确转换。凸轮应用于附属轴线遵循主设备的非线性属性的情况。凸轮操作的类型根据 Camming Mode 变化。
Read		读取来自轴线、坐标系、反馈及其它来源的状态及数据信息。使用读取方法来获得不同来源的信息。
Write		写出数据信息至轴线、坐标系、反馈及其它来源。使用写出方法来写出信息至不同来源。
Reset Position		重置指定轴线或坐标系上的位置。
Stop		停止轴线或坐标系上的当前运动。此操作的特性根据 Stop Mode 变化。

error out,
aborted,
active等
以下几方
用:

Power		启用及禁用指定轴线或坐标系源上的轴线和/或驱动。
Clear Faults		清除LabVIEW NI SoftMotion故障。

done,
busy, 及
输出按照
面发生作

使用LabVIEW NI SoftMotion功能块

以下技巧有助于你在LabVIEW中使用LabVIEW NI SoftMotion功能块编辑程序:

功能块:

- 无阻塞且一直在定义的时间段内执行。
- 触发后将命令传递至基于“execute”输入的上升沿运动管理器。
- 如果运动管理器通过“done”输出成功完成命令的任务，则提供反馈。
- 是副本化（具有唯一存储空间）并可重入的，但每个副本仅可从程序中的一个位置调用。
- 必须在LabVIEW Project中某部分的VI中执行。
- 具有双击对话框，可用于配置默认值，自动绑定数据至变量并将数据源配置为终端、变量或默认。
- 右击菜单可显示方法。例如，“Stop Move”功能块可能是减速、急停或驱动失效。
- 必须一直位于循环内。根据你的应用要求，你可以使用Wait Until Next ms Multiple Function对while循环定时或使用定时循环。

你应该使用功能块状态输入和输出（execute, done等）而非标准的LabVIEW编程方法，来决定功能块执行的顺序。例如，不要将功能块置于选择结构内，除非选择结构被状态输出控制。考虑以下框图：



图5.37. 使用运动功能块进行运动编程的错误实例

如果你不熟悉功能块，你可能认为这个代码会将轴1移动到1000的位置而后移回0。然而，这些功能块实际上不会执行运动——当它们发现“execute “输入上的上升沿时，仅仅会将命令发送至运动管理器。

相反，以上代码使驱动和轴线可用，但不会发生运动。

- Power函数发现了位于执行输入的上升沿（默认值为假，因此如果连线真值，首次迭代执行）。
- 当它接收到上升沿时，它将命令发送至运动管理器，时轴线和驱动可用。
- 这个无阻塞函数在继续LabVIEW代码前，不等待管理器完成命令，且“done “输出在首次迭代中为假值。

- 代码链中的剩余块没有发现位于执行输入的上升沿，也不会对管理器发送任何命令。
- 没有运动发生。

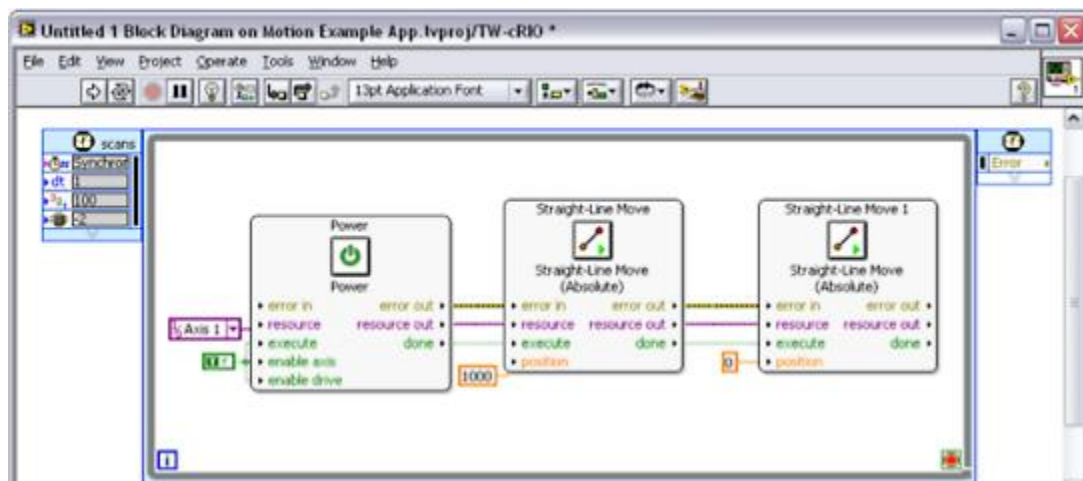


图5.38. 使用运动功能块的工作程序实例

本例中，驱动生效，移动至位置1000，然后移动至位置0，但它并未执行重复运动。这是因为每个功能块只接收一次执行输入上的上升沿。

为使轴线在位置1000和位置0之间重复循环，你需要编写图5.39所示的一段代码。

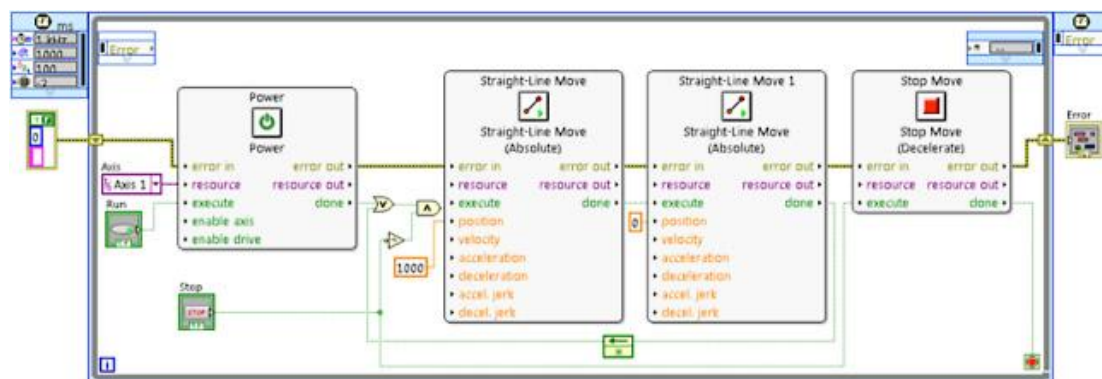


图5.39. 使用运动功能块的重复运动编程的正确示例

这段代码引起位置1000和位置0之间的重复运动。

- 带有Power功能块的循环发送命令至运动管理器。
- 然后检查管理器的命令状态。一旦得到命令完成的信息，它便将真值写入done比特并退出首次循环。
- 在定时循环中，第一个Straight-Line Move功能块发现位于执行终端真值，并命令管理器运动至位置1000。
- Done输出在运动完成前为假值，而第二个Straight-Line Move由于执行输入为假值并不运行。
- 在循环的每次迭代中，第一个Straight-Line Move检查管理器的动作是否完成。当首次运动完成时，它将在done终端上输出真值。
- 第二个Straight-Line Move发现位于执行比特的由低至高的转换，然后发送命令至管理器。
- 当此功能块收到完成命令的确认信息，它将done终端的假值变为真值。
- 通过位移寄存器，将使第一个函数功能块发现执行终端上的由低至高的转换并重复运行。
- 如果有停止命令被执行，带有停止命令的清除代码将使运动停止。因为运动管理器是一个单独的进程，如果没有停止命令发送给管理器，尽管LabVIEW代码不再执行，运动直到完成前仍将继续。

以下表格给出了运动功能块的概述。

LabVIEW示例代码



这一部分提供例子的LabVIEW 代码

状态机或状态图是运动应用的常用编程架构。然而，虽然状态机能够增加应用的灵活性，在状态机或状态图中使用运动功能块会产生一些额外的警告。

在顺序编程中，基于功能块**error**、**aborted**及**done**输出的运动命令间可以转换。（**busy**输出是这些其它状态的逻辑组合，因此对于简化的编程，你也可以监控**busy**输出。）在状态机编程中，可能会有平行的状态或事件影响运动，并使状态在运动完成前退出。

例如，你可能希望建立停止命令能被发送至系统的逻辑。如果停止状态与运动状态平行运行，运动管理器停止运动且运动状态在**aborted**输出上返回真值。

你也可以构建退出代码，而无需等待**abort**命令通过功能块返回，例如当你为程序编写**ESTOP**状态时。在大多数应用设计中，**ESTOP**立刻退出全部其它状态并转换至安全紧急停止状态。这种情况中，由于功能块没能彻底退出，你需要在执行前将其重置。当对执行输入写入假值时，功能块被重置。

为处理这一问题，你需要确保功能块有假值写入执行输入。在LabVIEW中你可以选择几种方法来执行此操作，包括为每个运动状态维护状态数据而添加逻辑，以确保功能块在执行前一直处于清除状态。或者你可以简单地在一个状态首次循环时将假值写入执行输入。请注意这将使你的运动执行延迟一个周期。

设想一个简单状态机的例子，它具有四种状态：**idle**、**initialize**、**move**及**stop**。在每种状态中，**busy**输出被用于确定功能块是否完成。

可以由任意状态转变为停止运动状态。这意味着停止状态可能离开运动，或在执行期间将状态初始化，并使功能块再次执行前需要重置。为达到此目的，状态机被设置，使每个功能块在首次迭代时接收执行输入上的假值，并在随后每次迭代时接收真值。这将自动清理被终止的功能块。

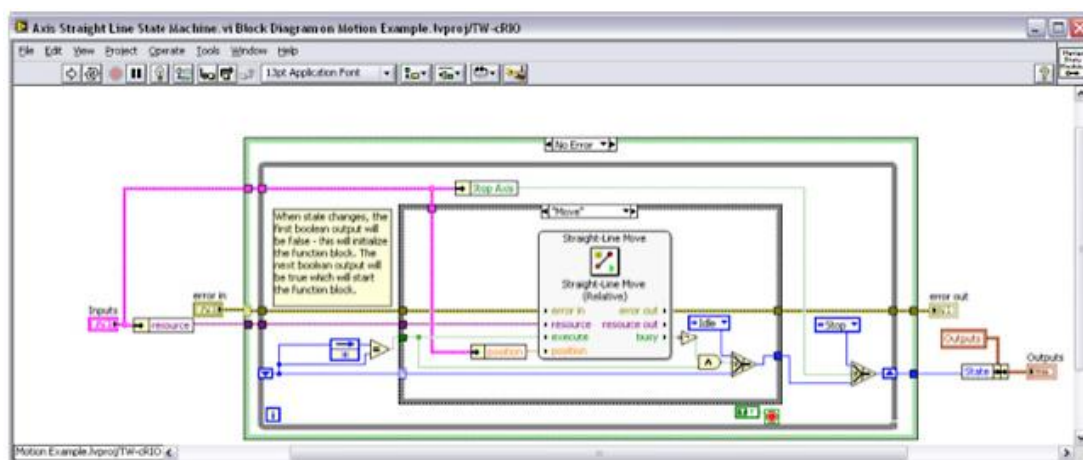


图5.40. 具有运动功能块的状态机

你可以在初始化和关机状态时，将此状态机放置在你的标准CompactRIO控制器架构中。

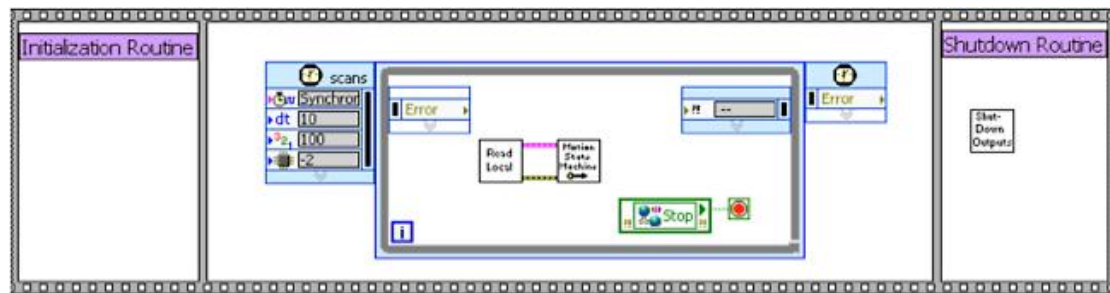


图5.41.具有运动命令的状态机可以放入标准控制器架构

第六章

通过LabVIEW FPGA定制硬件

通过LabVIEW FPGA扩展CompactRIO

上文讨论的控制结构主要研究使用CompactRIO的板载FPGA。它通过一个固定的特征来运行CompactRIO Scan Mode，从而达到使用CompactRIO的板载FPGA的目的。本章将研究在什么时间、因为什么原因以及怎样去扩展FPGA的功能。扫描模式支持大多数CompactRIO的C系列I/O模块以及多种高级功能，比如正交调幅编码器测量和PWM发生器。使用LabVIEW FPGA定制硬件之后，FPGA将拥有更多的功能。

通过用LabVIEW FPGA，可以实现以下功能：

- 以数万Hz的频率采集模拟波形
- 以高达40MHz的频率产生自定义的数字脉冲序列
- 执行自定义数字通信协议
- 以数万Hz的频率运行控制循环
- 使用扫描模式不支持的模块，包括CAN与PROFIBUS通信
- 实现自定义计时、触发、滤波

由于CompactRIO FPGA是完全可定制的，因此可以同时运行扫描模式和自定义LabVIEW FPGA。

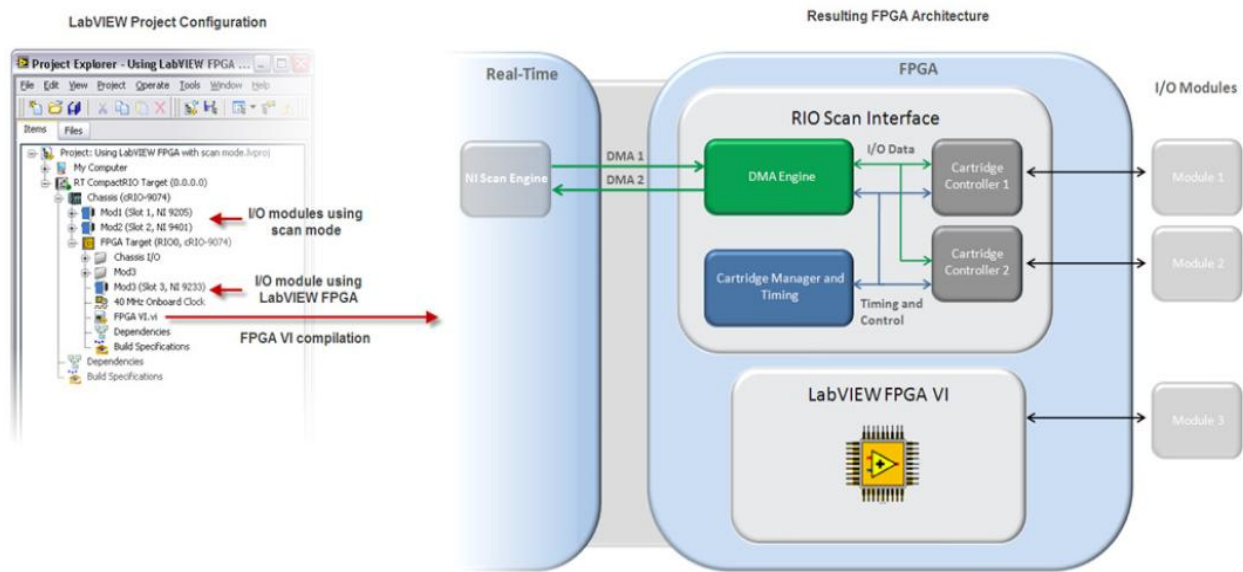


图6.1 使用自定义的LabVIEW FPGA扩展扫描模式的功能

使用LabVIEW FPGA的应用场合

由于种种原因，需要使用LabVIEW FPGA来扩展扫描模式。将CompactRIO上的实时处理器与可编程的FPGA结合，这样就可以创建一个能够利用各个计算平台优势的系统。实时处理器擅长浮点计算、分析以及外部通信，比如网络发布的共享变量与网络服务。

FPGA擅长处理需要超高速逻辑运算和精确计时的小型任务。下面列出了一系列需要直接编写FPGA的情况。

高速波形的采集/生成（高于1kHz）

CompactRIO Scan Mode最适用于运行速率低于1kHz的控制循环，但很多C系列I/O模块能够以更高频率采集和生成波形。如果想充分利用这些模块的所有特征，并以高于1kHz的速率采集或生成波形，就可以使用LabVIEW FPGA以自定义的速率采集波形，这样就能满足要求。

自定义触发/计时/同步

使用可重新配置的FPGA，就可以创建简单的、高级的，或其他自定义的触发、定时方案以及I/O或机箱同步。这些操作可能非常复杂，就像根据采集到的模拟波形是否超过阈值来触发一个自定义CAN信息，也可能很简单，就像采集外部时钟源的上升边缘上的数据。

基于硬件的分析/生成与协同处理

许多传感器在应用时要输出大量的数据，使得实时处理器不能及时有效处理地处理这些数据。可以把FPGA作为一个有效的协处理器来分析或生成复杂的信号，这样就可以释放处理器来处理其他重要的线程。这种基于FPGA的协同处理器经常被用于像以下列举的应用程序中：

- 编码/解码传感器
 - 流速计
 - 标准或自定义数字协议
- 信号处理与分析
 - 谱分析（快速傅里叶变换与窗口）
 - 滤波、求平均值等
 - 数据简化
 - 第三方IP整合
- 传感器仿真
 - 凸轮和曲轴
 - 线性微分变压器（LVDTs）
- 硬件在环仿真

最高性能的控制

不仅可以使FPGA来高速采集与生成波形，还可以在FPGA上执行许多控制算法。可以使用多通道单点I/O，可调PID或其他控制算法以高达几万Hz的循环速率来执行确定性控制。

不支持的模块

扫描模式不支持一些C系列模块。对于这些模块，需要用LabVIEW FPGA在I/O与实时应用程序之间建立接口。扫描模式支持的模块详见“[CompactRIO Scan Mode支持的C系列模块](#)”。

不支持的目标

拥有1M 门FPGA的CompactRIO不能完全支持扫描模式。在扫描模式不支持的目标上也可以执行一些扫描模式的功能，但必须使用LabVIEW FPGA。在“知识基础”里的文章“使用CompactRIO Scan Mode及其不支持的目标”中描述了如何用LabVIEW FPGA在扫描与其不支持的FPGA目标之间建立一个自定义扫描模式界面。

FPGA概述

FPGA即现场可编程门阵列，是由三个基本部分组成的可编程芯片：逻辑模块、可重构的内部连线和I/O模块。

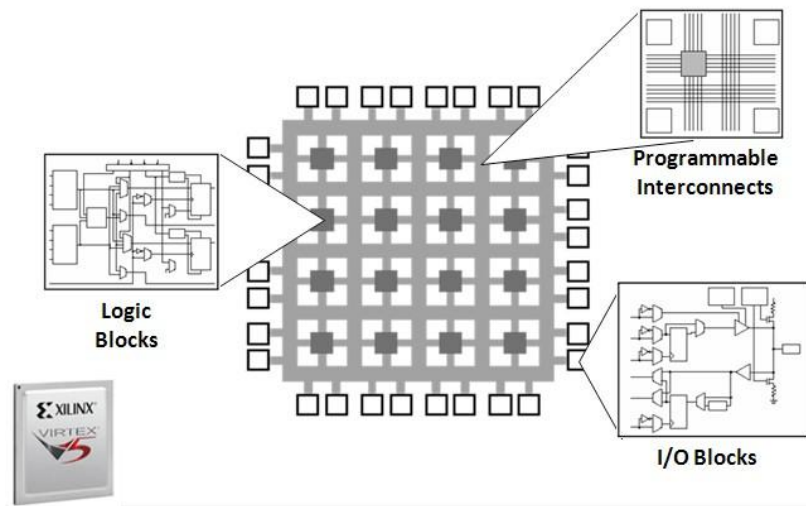


图6.2一个FPGA由可配置逻辑模块、I/O模块以及可重构的内被连线组成

逻辑块是由很多数字部件组成的，比如查找表、乘法器、多路转换器，由它们处理数字以得到所需要的结果。可重构的内部连线把这些逻辑块相互联系起来。可重构的内部连线是一种在不同逻辑块之间传递信号的微型电路板。可重构的内部连线也将信号传送到I/O模块，I/O模块与芯片上的插脚相连，与周围的电路交换数据。FPGA就是一个空白的硅板，你可将它编制成任何自定义的数字硬件。对这些FPGA芯片进行编程是很困难的，通常由娴熟的编程师和硬件工程师来完成这项工作。National Instruments公司使用LabVIEW FPGA通过图像化系统设计将这些设备的编程抽象成图像，这样几乎每个人都可以充分利用这些强大的可重构的芯片了。

LabVIEW FPGA VI被合成了到物理硬件门上，这些物理硬件门通过可重构的内部连线连接起来。这样FPGA的编译过程就与传统Windows上的LabVIEW或LabVIEW 实时应用程序不同。当向FPGA写入代码时，同时也将这些相同的LabVIEW代码写入到了其他目标上，但运行后，LabVIEW则通过不同的进程来运行这些代码。首先，FPGA生成VHDL代码，并将其传递到Xilinx编译器上。然后，Xilinx编译器合成这些VHDL代码，并将这些代码放置在一个位文件里。最后，将位文件下载到FPGA上，由FPGA负责处理这些代码。这个过程比其它LabVIEW编译要复杂得多，可能需要5至10分钟来完成这个过程。对于更复杂的代码，这个过程可能会花费几个小时。由于编译时间相对较长，所以在编译之前需要花费更多的时间来调试和优化LabVIEW FPGA代码。

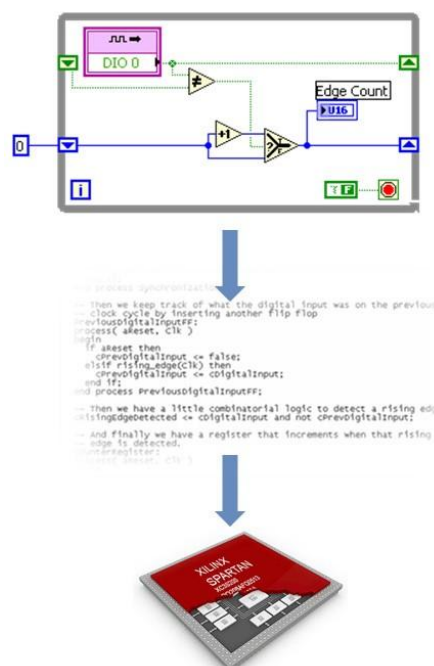


图6.3 LabVIEW FPGA编译器将代码转换成VHDL，然后使用使用Xilinx工具生成一个位文件，这个位文件能在FPGA上直接运行

FPGA的优点

因为LabVIEW FPGA代码能直接在硬件上运行，因此可以充分利用以下基于FPGA设计的优点。

高可靠性

在FPGA上运行的LabVIEW FPGA代码是非常可靠的，因为逻辑被编译到了物理硬件设计上。这样物理硬件就变成了一个非常可靠的硬件芯片。

高确定性

基于处理器的系统通常包括几个抽象层，它们帮助在多个进程之间协调任务以及共享资源。驱动层控制硬件资源，操作系统管理内存与处理器带宽。对于任何给定的处理器，一次只能执行一条指令。当使用一个优秀的优先级层次进行编程时，实时操作系统优能将抖动降低到一个有限的范围内。FPGA不使用任何操作系统，这样就使影响并行执行和确定性硬件的可靠性因素降到了最低。使用基于NI FPGA的硬件，可以使关键部件获得高达25ns的精确定时。

真正的并行运行

多线程应用程序被分解成多个平行的代码段，然后以循环的方式来执行这些代码段，这样看起来多个程序就在并行执行。多核处理器扩展了这个概念，它允许多个应用程序同时真正地并行执行多个代码。但处理器的核数限制了同时执行的并行代码的个数。FPGA在硬件中以平行循环方式的执行并行代码，所以不会受到处理器核数的限制。在整个FPGA中的每一段平行代码都可以同时执行。甚至通过在FPGA上以流水线的方式来执行程序，传统的串行运算也能提高其处理能力。

可重构性

因为FPGA芯片是可以重构的，所以你将来需要的任何修改，它都能满足。作为一个产品或成熟的系统，你不必花费时间重新设计硬件或修改面板布局，就可以提高其性能。这对工业通信协议来说尤其适用。随着通信协议的发展和进步，可以在FPGA中修改协议的执行方式，来支持最新的科技特性和变化。

即时开机

由于不依靠任何操作系统，LabVIEW FPGA代码就可以直接在FPGA上运行，所以下载到FPGA闪存上的代码，在CompactRIO机箱通电几毫秒内就可以开始运行。这个代码可以执行控制循环或设置启动输出值。

用LabVIEW FPGA编程

当需要扩展CompactRIO硬件上的控制结构使其包含自定义的FPGA编程，就可以考虑CompactRIO的三种编程模式：扫描模式、完全FPGA接口和混合模式。控制结构的说明文件绝大部分都在讨论使用CompactRIO扫描模式来检索I/O。从前文已经了解到，这种编程技术是FPGA的固有特性，它通过FPGA整理数据并将这些数据直接传递到实时主机上。这一部分详述了如何使用FPGA在混合模式下直接访问I/O，同时使其它I/O处于扫描模式。在混合模式下，可以使用FPGA编程模块来处理缓冲波形的采集、内处理以及某些扫描模式不支持的特定模块。

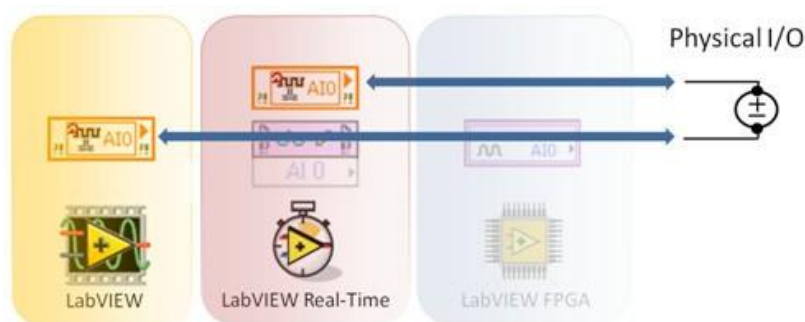


图6.4 在扫描模式下，可以通过扫描模式或者FPGA接口来采集物理I/O

CompactRIO上的混合模式

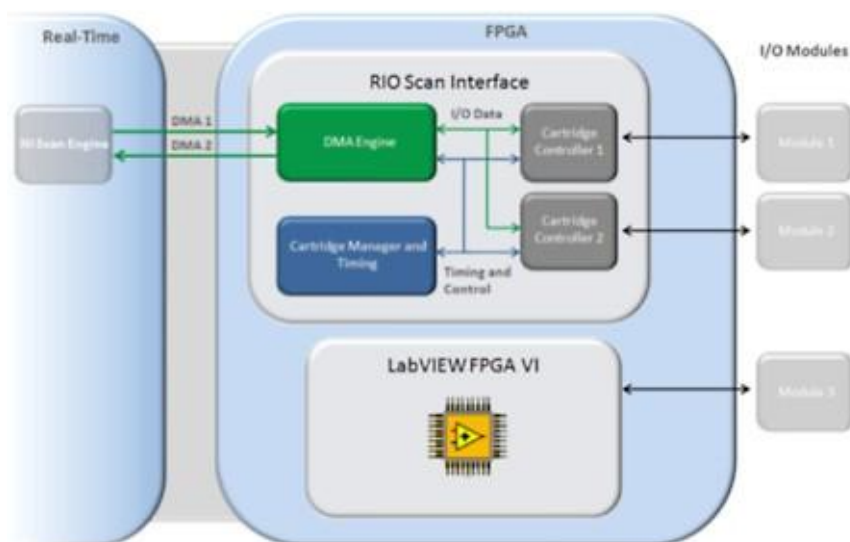


图6.5激活混合模式后，使用这个模块将一个FPGA VI写入接口并将数据传递到实时主机上

在混合模式下，当在FPGA上对一些模块进行编程时，仍然可以使用其他模块上的NI RIO Scan Interface。通过将模块工程项目从CompactRIO机箱下拖放到FPGA目标下，就可以激活一个特定模块的FPGA程序设计。这样就可以为与自定义代码编写FPGA了，这些自定义代码与其他模块的扫描接口并行运行。

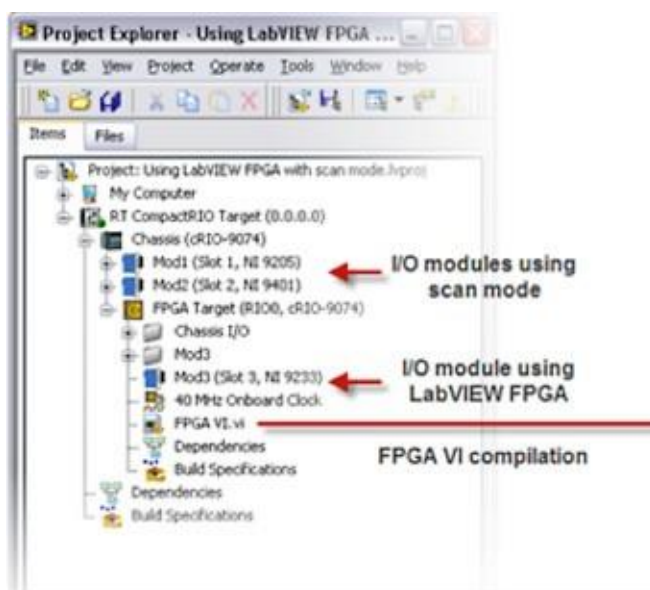


图6.6这个混合模式里的工程展示了CompactRIO机箱下的一些模块和FPGA下的其他模块（图中为模块3）

使用混合模式，需要拥有FPGA编程与实时目标上FPGA接口的应用知识，因为通常需要混合模式的应用程序。从本质上讲，将更加复杂。在FPGA与实时目标之间有两种传递数据的方式：主机接口和用户自定义的I/O变量。主机接口是低级接口，它提拥有更高的灵活性，但同时操作也更加困难。用于自定义I/O变量允许FPGA直接向实时存储列表读写单点值，但只能传送扫描模式支持的数据类型。

下面将介绍五个应用程序的例子：

1. FPGA的异步内处理：使用主机接口，将单点 I/O值传递到实时应用程序上。
2. FPGA的同步内处理，使用主机接口，将单点 I/O值传递到实时应用程序上。
3. FPGA的异步内处理，使用自定义的I/O变量，将单点 I/O值传递到实时应用程序上。
4. FPGA的同步内处理，使用自定义的I/O变量，将单点 I/O值传递到实时应用程序上。

5. 使用主机接口，实现FPGA与实时应用程序之间的高速数据流传递。

例子—使用主机接口进行单点 FPGA/Real-time通信



这一部分提供例子的LabVIEW 代码

FPGA编程基础

扫描是一个采集速率高达1kHz的单点采集方法。然而，大多数的模块能够达到更高的采样速率。为使采集模块的输入速率高于1kHz，需要完成下列步骤：

1. 通过将模块的FPGA编程放置在工程的FPGA环境中，来激活它。（上一节讨论过）
2. 编写FPGA VI：
 - a. 从FPGA I/O节点上读取数据
 - b. 利用LabVIEW结构和逻辑来定义采样率
 - c. 检查错误状态来保证数据的完整性
 - d. 可以在本地处理数据，也可以将数据打包传入DMA FIFO由主机处理。
3. 编写一个主VI，既可以读取DMA FIFO上从数据包，也可以读取FPGA上任一内处理函数上的结果。

从I/O 节点上读取数据

创建一个新的FPGA VI，右击项目中的FPGA Target，并选择New» VI。对于FPGA Target下的每个模块，都会看到一个相应的文件夹，文件夹里包含相应模块的所有的I/O points（如果没有，右击该模块，选择New»FPGA I/O并添加所有可用的资源）。将在一个I/O点拖放到程序框图上，就得到一个I/O节点。

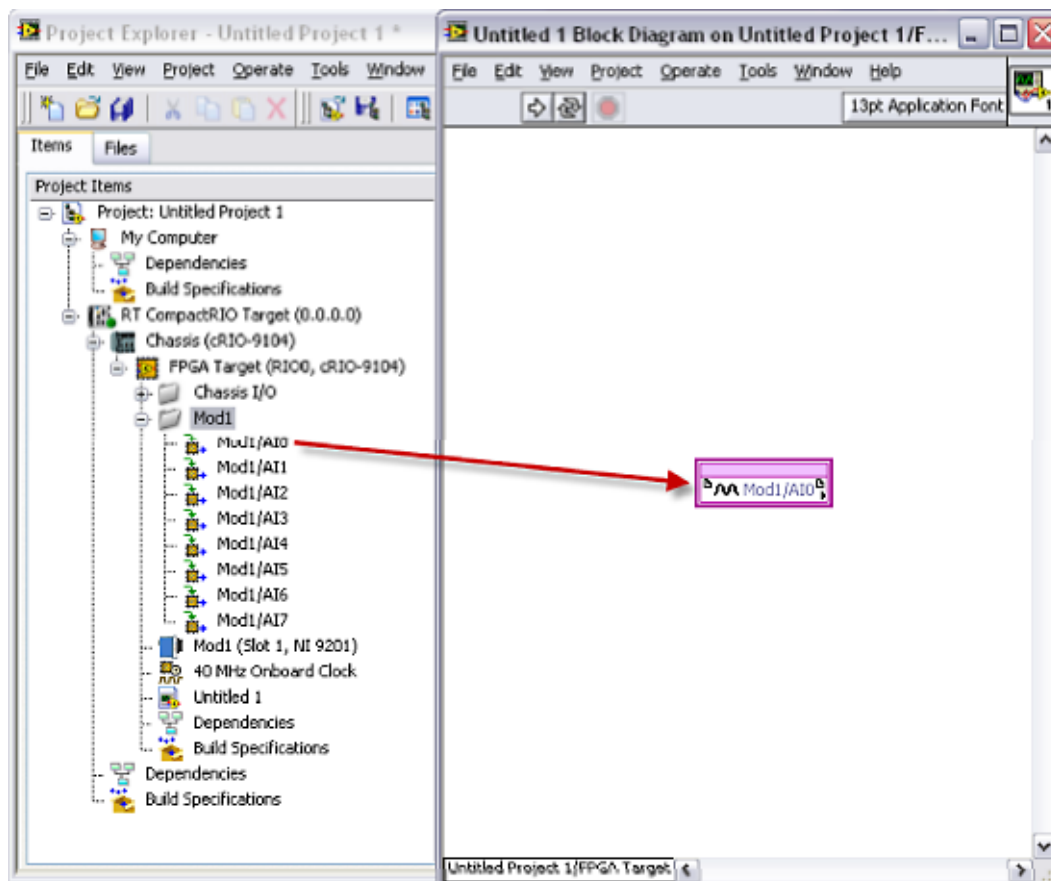


图6.7给FPGA程序框图上拖放一个I/O节点

运行VI，开始进行编译。在开发程序的过程中，可能会使用FPGA仿真。要将FPGA Target转换为仿真，右击目标并选择**Execute VI on»Development Computer with Simulated I/O**。

利用LabVIEW结构和逻辑来设置采集率与触发

FPGA I/O 节点返回每个通道的单点数据。为了获得所需的采样率，把FPGA I/O 节点放入一个循环，并用循环定时器控制采样周期。使用单层顺序结构来确保在其他程序运行之前，首先调用循环计时器，比如FPGA I/O 节点。

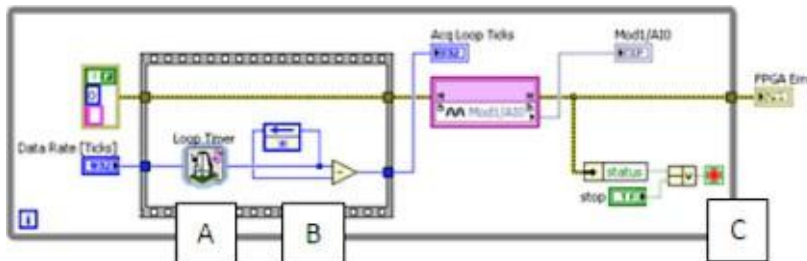


图6.8从一个模拟输入模块采样

接下来，将解释个例子中的代码是如何运行的：

- 首先调用循环定时器，记录下开始运行的时间并立即执行后面的程序。当内部时钟达到设定的时间段后，再度调用循环定时器，然后执行剩余的代码。如果循环内代码的执行时间超过了设定的循环周期，循环定时器就会立即返回并使用当前的值作为下一次循环的开始时间。使用循环定时器的计数端子来设定循环的采样周期。计数端子根据FPGA时钟来计时，FPGA时钟的默认运行速率为40MHz。比如FPGA时钟运行20000下，采集一个数据，那么采样频率就是2KHz。
- 循环定时器确保了While循环的运行速率不会超过设定的频率；但没有给出While循环的运行速率低于设定的频率时，如何处理。为了开发FPGA代码的衡量标准，最后的办法就是测量循环周期来确保以正确的速率采集数据。可以使用循环定时器的反馈节点减去连续循环的计时时间，就可以达到这个目的。
- 错误检查机制是确保数据完整型的关键环节。如果与C系列模块的通讯信息丢失（移除模块或者出现致命错误），那么FPGA I/O节点上的错误簇就会停止采集数据。错误信息也会通过“FPGA Error”指示器传递到主机应用程序上。当然也允许探测其他的错误条件。

FPGA的内处理

根据应用程序的需求，实时主机也许只需要对采集到的数据进行某个处理。这样有时候有利于减少数据处理任务的负担（过滤、求平均值或者测量），比如可以使用FPGA来采集数据、执行RMS运算以及将结果传递到实时控制器上。

下面的FPGA代码从模块1的0通道读取数据，通过数据线传递数据，然后进行RMS计算，最后通过一个叫“RMS result ch0”的FPGA指示器将结果传送到实时程序里。

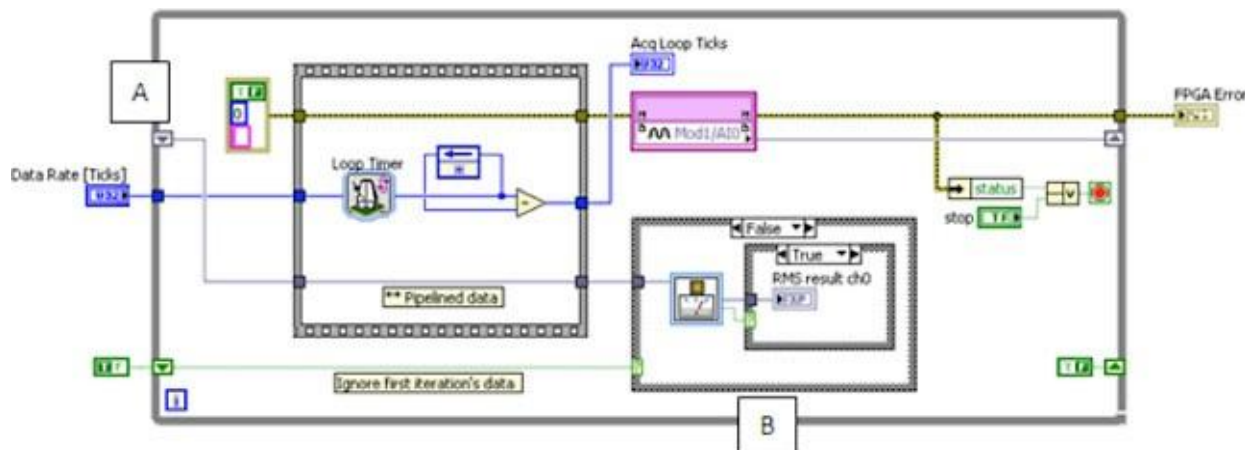


图6.9 将数据传递套内处理程序上

接下来，将解释个例子中的代码是如何运行的：

- A. 执行FPGA I/O 节点会占用一部分采样周期，尤其是在高采样率下运行的时，所占用的时间将更长。因此，使用I/O节点进行串联内处理，就可能发现在指定的采样周期内不能完成循环。因为FPGA可以很容易平行运行，所以最好的办法就是将采集的样本点传递到移位寄存器，由下一个循环迭代去处理。这种技术，即流水线，使得我们可以平行处理采集的数据，并且当模块采样速度低于需求时，它能将模块下溢的风险降到最小。
- B. 因为采集的数据直到下一个循环时才会被处理，所以第一次循环里并没有有效的数据。一个选择结构使得首次内处理失效，并忽略了首次循环里的数据。随后的循环将对流水线传来的数据进行内处理。

检测错误状态：模块下溢

如先前讨论，模块下溢是指一或多个采样周期超过需求的周期。如果内处理时间比需求的采样周期长，或户设定了不支持的采样率，就会发生模块下溢。在运行过程中监视下溢状态的循环速率是极其重要的。

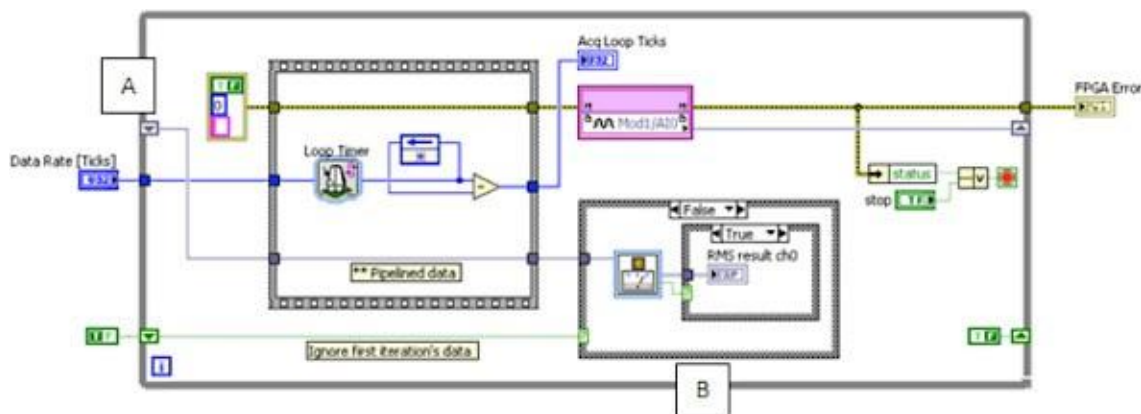


图6.10检查FPGA上的模块下溢

接下来，将解释个例子中的代码是如何运行的：

- A. 通过向需求的循环周期增加一些定时点，这样就建立了一个下溢临界值。实际循环周期会以此作为比较标准。实际循环周期同时也显示在FPGA指示器上，供给实时主机用查看。
- B. 可能需要花费几个循环来清除循环定时器的移位寄存器与内部C系列流水线，所以直到第二个循环完成，再开始检查下溢状态。
- C. 一个模块的下溢状态并不会停止采集循环，它只是简单地锁定一个布尔指示器。实时应用检查FPGA指示器，并决定采取合适的操作。因为FPGA更新“模块下溢”指示器要的速率比实时应用程序的检查速率快，所以用反馈节点和OR门锁定下溢状态是很重要的。通过锁定下溢值，就能保证如果采集过程中出现了下溢，指示器就会一直显示TRUE，这样实时应用程序就不会漏掉这个警报。

在实时程序中的主机接口与LabVIEW FPGA通信

实时程序需要使用FPGA传递单点数据与数据流以及设置与清除中断。使用用FPGA 主机接口就能实现这些功能。



图6.11FPGA接口函数面板

打开FPGA VI 引用

打开一个FPGA VI或指定的位文件和FPGA目标的引用。右击Open FPGA VI Reference函数，从快捷菜单中选择Configure Open FPGA VI Reference，就会显示Configure Open FPGA VI Reference对话框。从对话框中选择FPGA VI。也可以配置是否要FPGA VI

自动运行。

读取/写入控制

向FPGA目标所在的VI的输入控件或显示控件写入或读取数值。可以是一个触发条件、采样速率或其他由指示器或控制器设定的数值或参数。

调用方法

从FPGA VI的主VI里调用FPGA接口方法或操作。用这些方法来实现以下操作：下载、中止、重置以及运行FPGA目标上的 VI；等待并识别FPGA VI中断；读取DMA FIFO;写入DMA FIFO。根据目标硬件以及FPGA来选择不同的方法。必须连接FPGA VI Reference In 输入来检查快捷菜单的可用方法。

关闭FPGA VI 引用

关闭FPGA VI的引用，并选择地重置或中止VI的执行。

在实时代码中的初始化程序中打开FPGA文件的引用。配置FPGA VI使其自动运行。这样就将FPGA 位文件下载到了FPGA上，并开始执行。因为FPGA 位文件既包括LabVIEW FPGA 代码，也包含扫描模式中模块的代。这个VI程序必须能与系统中的任何I/O通信。

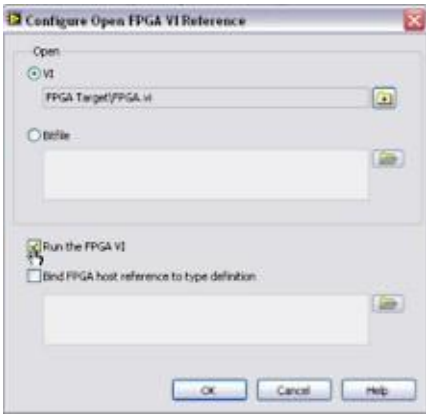


图6.12使用对话框配置Open FPGA VI Reference,可以选择VI或者位文件并进行配置使其调用时即执行

然后使用一个定时循环来创建一个FPGA Comm Task。在这个循环中，读取FPGA VI 前面板项目。执行任何必要的错误检查，如果数据有效，利用可以实时FIFO的单进程共享变量，把数据传到内存中。

最后在关闭程序中，关闭FPGA VI 的引用。一定要确保配置程序框图，使FPGA VI一直运行。如果停止运行FPGA VI，它也将停止扫描模式中模块的I/O，并使用关闭程序中的在其它设置产生竞态条件。

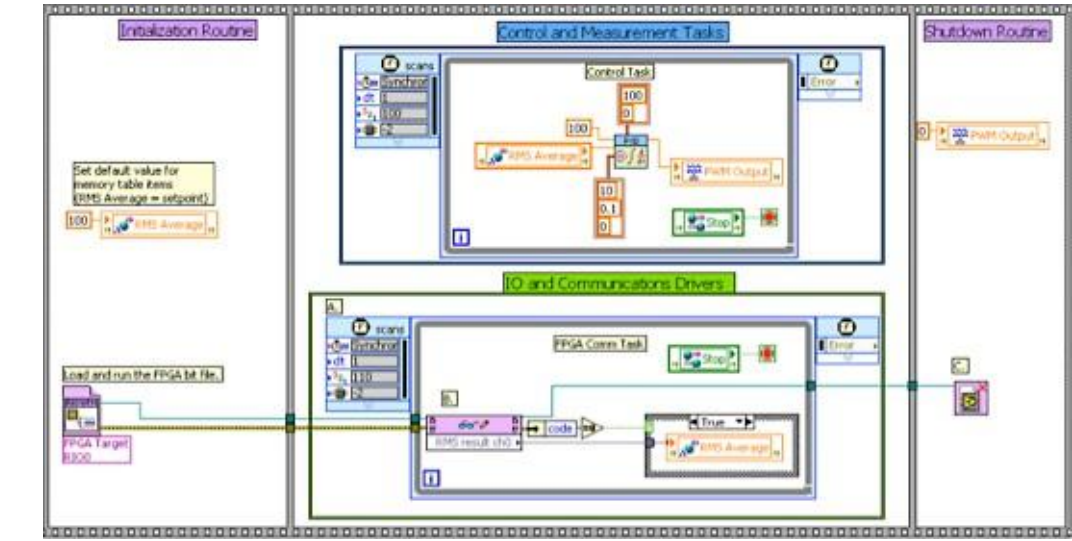


图6.13一个完整的实时应用程序，将单点变量传递到FPGA程序中并将数据插入到内存列表中

例子— 同步单点FPGA实时通信

对于某些控制应用程序，可能需要利用实时处理器上的循环来同步FPGA上的数据采集与分析。这样就能提供一致的I/O延迟同时也能减少抖动。使用中断是一个简单而有效地同步FPGA与实时代码的方法。

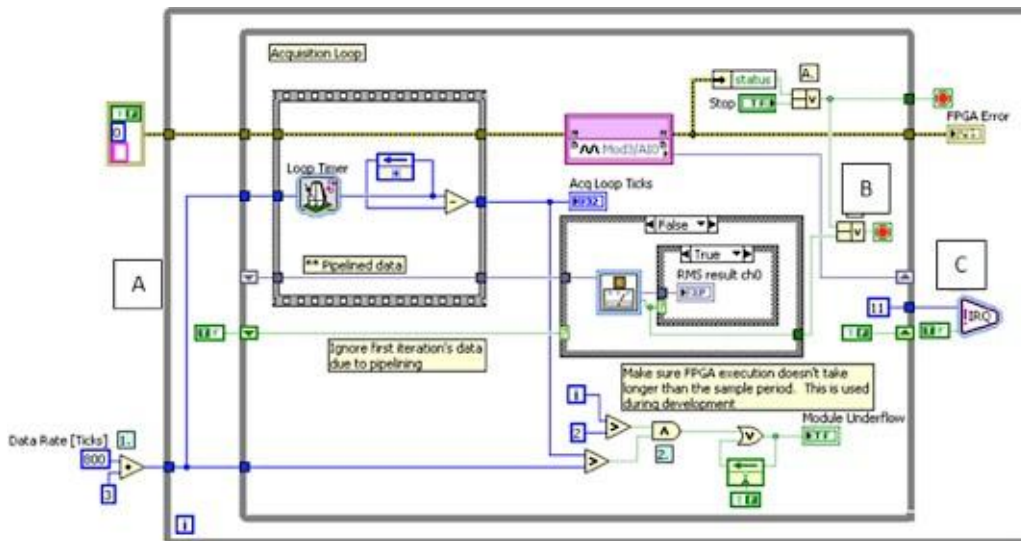


图6.14在LabVIEW FPGA上，可以设置并等待实时控制器的中断

通过实时应用程序控制循环，可以用一个简单的中断来同步FPGA的数据采集。

接下来，将解释个例子中的代码是如何运行的：

- A. 每个分析程序根据中断信号，都有一个内循环来控制波形采样速率以及一个外循环来控制计时。
- B. 一旦从RMS分析算法得到一个有效的结果，内循环将退出。
- C. 外循环在实时控制器上设置一个中断信号，并等待实时应用程序识别。

在实时应用程序中，可以更改FPGA Comm Task来处理中断。实时应用程序将等待FPGA传来的中断，以确得到最新的数据。然后读取这些数据并确认中断，来触发FPGA去测量与计算新的数值。

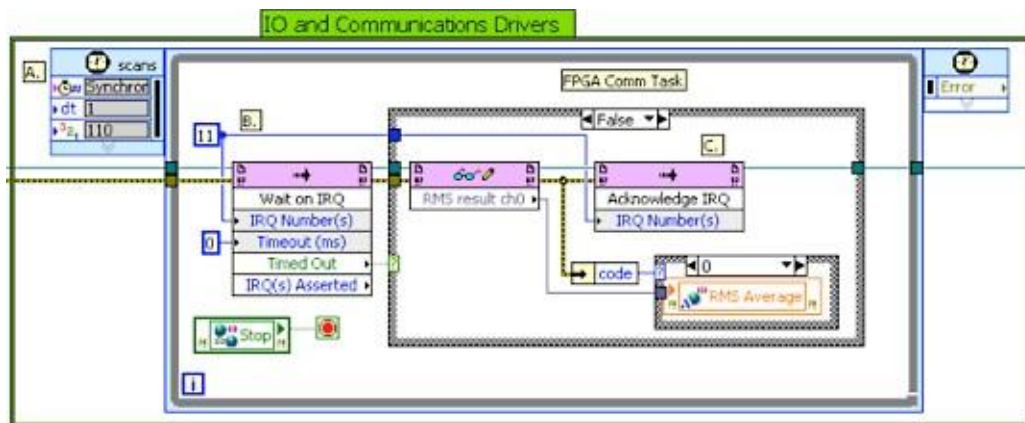


图6.15在实时程序中，等待中断、读取数据并确认中断

例子—使用自定义I/O变量进行简单的单点FPGA实时通信

自定义I/O变量

除了使用主机界面，还可以使用扫描引擎中的自定义I/O变量，在FPGA VI与实时主机VI之间传递数据。必须创建一些程序代码来在FPGA和实时应用程序之间访问以及传递连续的数据，使用NI 扫描引擎可以减少这些程序代码的数量。使用NI 9144确定性扩展I/O目

标时，自定义变量是唯一能通过网络传递自定义FPGA代码的机制。

创建自定义I/O变量

右击Project Explorer窗口中的机箱项目，选择快捷菜单中的New>User-Defined Variable来创建一个新的I/O变量。因为所有I/O变量都是单向的，所以必须配置每个自定义I/O变量的方向，可以是FPGA to Host，也可以是Host to FPGA。

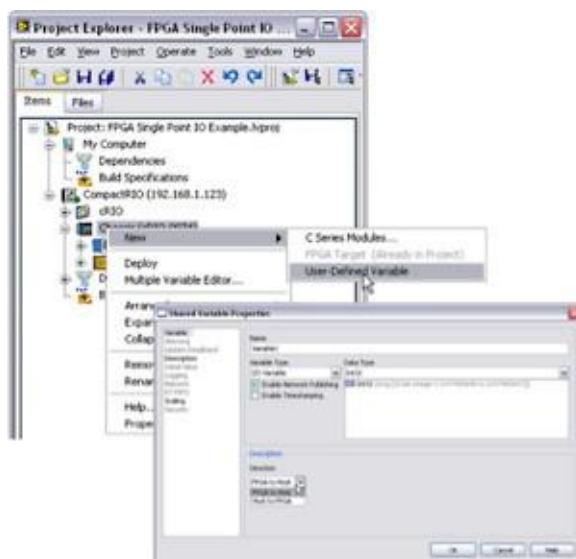


图6.16使用自定义变量在FPGA和实时程序之间通讯

在FPGA程序中，除了使用前面板控制器和指示器外，还可以把I/O变量拖动到图标中。

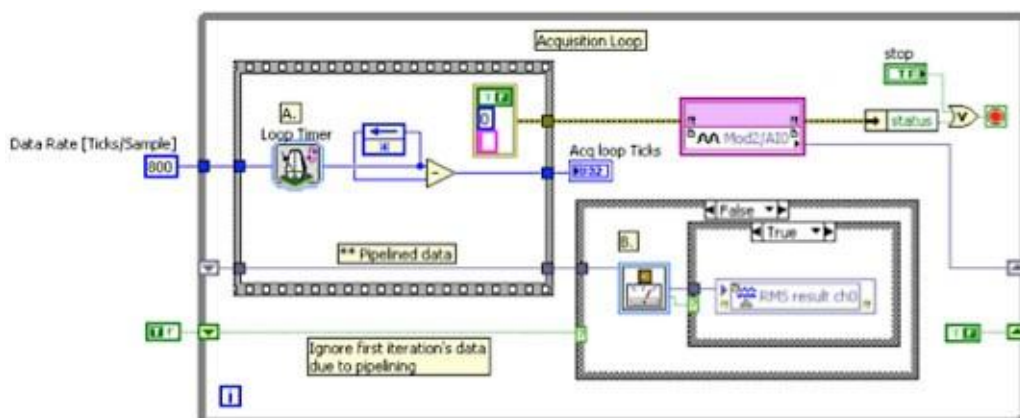


图6.17在FPGA程序中，可以直接向自定义变量写入数据

I/O变量被当做扫描的一部分来读取，并被自动加入到实时控制器的内存列表中。在实时代码中，可以忽略FPGA通信循环，直接从主控制循环中读取I/O变量。仍然需要加载和关闭FPGA代码的引用。

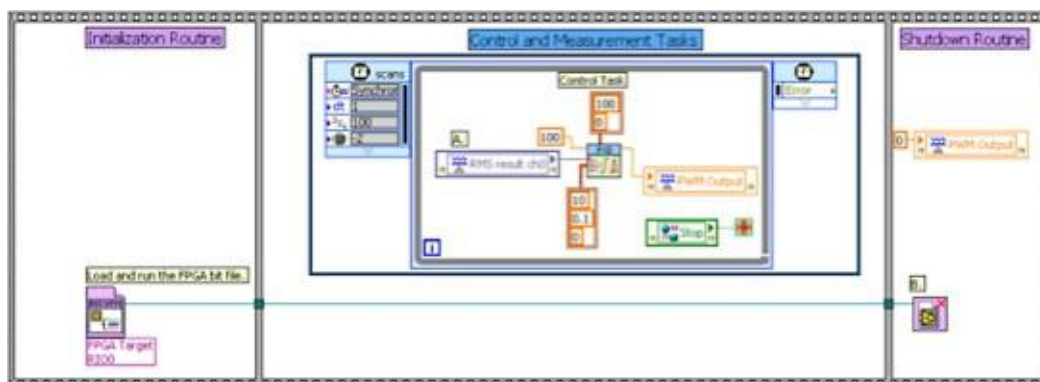


图6.18变量被自动扫描到内存列表中，这样就可以直接从控制程序中读取变量值

例子—利用自定义I/O变量实现同步FPGA/实时通信

扫描引擎也为FPGA提供计时信息。通过向FPGA VI增加Scan Clock I/O项目，来访问计时信息。这个I/O项目将计时信息从扫描引擎传递到FPGA VI，比如高频信号中FPGA时钟周期的数量。使用计时信息设计一个程序，来保证在FPGA VI与实时主机VI之间传递的数据是一致的。用Scan Clock I/O 项目、方法和属性来监测扫描引擎的信息。

Scan Clock I/O 项目

使用FPGA I/O 节点来访问Scan Clock I/O 项目。将Scan Clock I/O 项目从工程的Chassis I/O文件夹中拖动至FPGA VI程序框图里，或将FPGA I/O节点放置在程序框图上，点击FPGA I/O 节点的元素部分，并从快捷菜单中选择Chassis I/O»Scan Clock。当扫描引擎没有在FPGA VI 和实时主机VI间传送数据时，Scan Clock I/O状态为TRUE，这时可以安全地向I/O变量写入数值。如果在时钟错误的情况下写入数值，虽不影响判定，但不能保证下次扫描时能够成功传递该数据。

使用FPGA I/O Method Node等待扫描时钟的上升边沿。这样，一旦扫描引擎停止传送数据，就可以立即写入FPGA代码来开始运行。

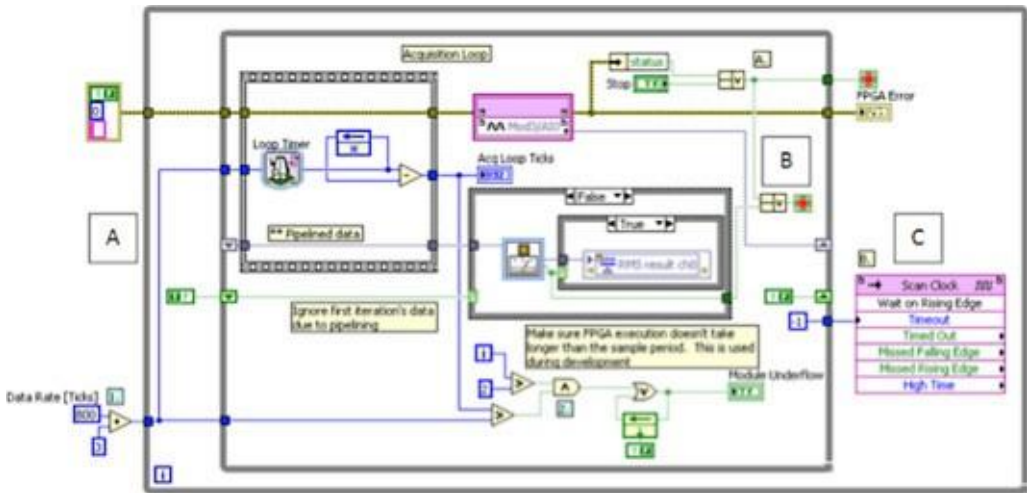


图6.19在LabVIEW FPGA中，等待扫描时钟的上升边沿来同步FPGA和实时程序

配置FPGA代码使其等待时钟的上升边沿来同步FPGA，这样主机就能随时使用新数据。接下来，将解释个例子中的代码是如何运行的：

- A. 每个分析程序根据中断信号，都有一个内循环来控制波形采样速率以及一个外循环来控制 计时。
- B. 一旦从RMS分析算法得到一个有效的结果，内循环将退出。
- C. 外循环等待扫描时钟的上升边沿

实时应用程序不需要任何修改，因为扫描引擎已经运行并开始为控制循环计时。

自定义I/O变量的附加说明：

- 可以只在与扫描引擎一起工作的FPGA目标上执行自定义 I/O变量。参考目标硬件的详细说明来了解扫描引擎的支持信息。
- 自定义I/O变量不支持仿真模拟。为了改变一个FPGA VI的执行目标，右击包含该VI的机箱项目，并选择Execute VI on»FPGA Target。
- 执行自定义I/O变量，使其只在FPGA VI和运行在同一个机箱上的实时 VI之间进行通信。然而，如果自定义I/O变量能够使用网络发布的共享变量，就可以在同一LabVIEW工程中的任何一个实时VI或基于Windows的VI中使用该变量。例如，可以使用网络发布的I/O变量来创建一个在Windows中运行的用户界面。

例子—利用DMA FIFO采集波形

配置FPGA与实时硬件之间的通信

可以使用DMA来在FPGA与实时硬件之间传递高速数据。为传递中的数据创建一个DMA缓冲区，右击FPGA目标，选择New...»FIFO

给FIFO结构起一个解释性的名字（比如DataU32），并选择target-to-host作为其传递类型。还可以设置数据类型以及FPGA FIFO的长度。数据将自动从FIFO中传递到实时控制器RAM中的数据缓冲区。FPGA FIFO缓冲区的长度并不像实时数据缓冲区的大小以及用途那样重要。点击对话框中的OK键来向工程中添加新的FIFO，然后可以将它拖放到FPGA的程序框图中。

这样就得到了四个数据通道，而不只是进行把数据放置在DMA FIFO中的内处理。主应用程序负责检索数据处理或传送。许多概念与内处理中用的例子相同。主要有以下几点：

- 循环计时器在第一次循环时产生一个时间标签，然后等待一个设定的时间段再执行第二个循环，以达到设定的速率。如果FPGA执行时间超出了设定的循环时间，就会为接下来的循环产生一个新的引用。可以将移位寄存器置于循环计时器之后，来测量实际循环速度。检验循环速率，确保以设定的速率采集数据。
- 使用错误检查机制确保数据的完整性。如果FPGA丢失了与C Series模块的通讯信息，I/O Node的错误簇会报告这个丢失信息。
- 运行FPGA I/O Node将占据一部分采样时间，特别是高速采样时，占据的时间将更长。数据流将数据转化放置在一个与FPGA I/O Node并行运行的代码里，这就减小了数据下溢的风险（也就是说，模块采样比要求的慢了）。
- 因为采用数据流，所以第一次循环或数据点是无效的。外部条件结构将丢弃第一个数据点。

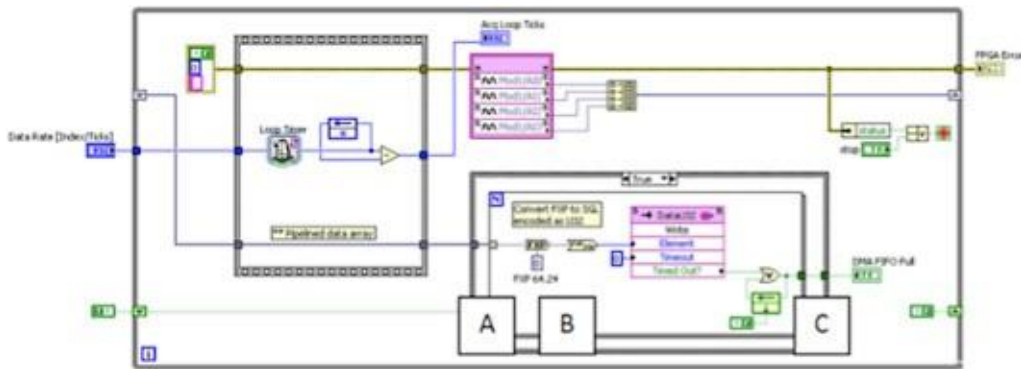


图6.20 将采集到的数据放在DMA FIFO中

接下来，将解释个例子中的代码是如何运行的：

- A. 本例主要研究从多通道获取数据。转换每个点的数据类型，并将各点放入到DMA FIFO中，最终会在FIFO中得到交错数据。
- B. 需要将每个数据点从定点型（FXP）转化为单精度浮点型（SGL）。执行转化的一个方法是通过DMA FIFO发送定点数据，然后在实时应用程序中将数据转化为单精度浮点型。一个更有效的方法是，在FPGA上把数据类型转化为单精度浮点型，并将其作为U32来编译。在实时应用程序中，将U32转化成单精度浮点型要比将定点型转为单精度浮点型快近40%。这个简单的技术节省了主控制器上宝贵的CPU资源。访问ni.com，可以获得更多关于FPGA上FXP-to-SGL的转化信息，以及最新下载资源和文档。
- C. 这个例子也检查了DMA FIFO是否下溢。如果DMA I/O Node发生超时，DMA FIFO中的数据就会溢出。如果发生这种情况，就会产生一个错误表示来通知主应用程序。

模块下溢与不同采样模式的支持

可以扩展上文的程序代码使其能检查模块下溢，以及支持多种采集模式。模块下溢检查与之前讨论的内处理中的例子相同。仍然设置一个临界值来显示模块采样率是否低于设定值，并在报告下溢之前给FPGA循环几次，来清空数据流和移位寄存器。

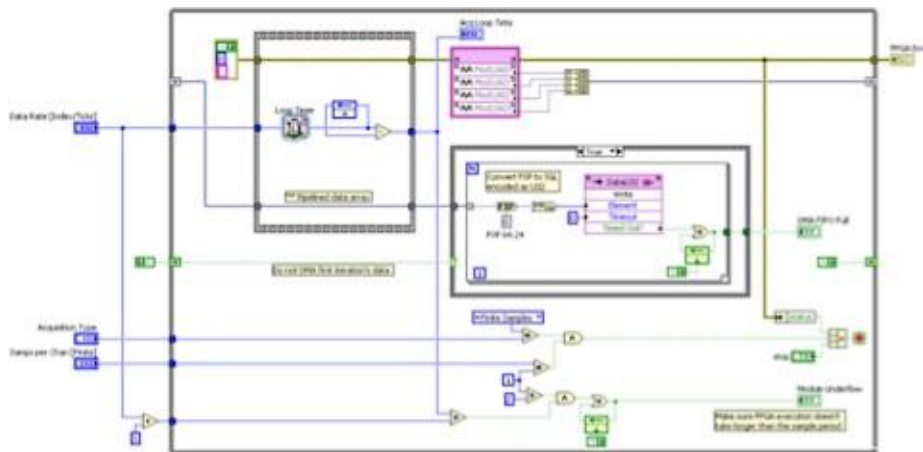


图6.21 下溢检查以及采样模式

本例包含两个新的控制：采集类型和每通道采样（有限）。如果选择了有限采集，那么每个通道读取了指定数量的数据后，采集循环就会自动停止。采集类型控制使得实时应用程更加灵活，可以在连续或有限采集之间进行切换，无需重新编译FPGA代码。采集类型也被用来决定给主控制器上的实时DMA缓冲区分配多少内存。

主机同步与自动重启

进行连续DMA传输时，同步FPGA与主机应用程序就显得尤其重要。如果FPGA在实时应用程序准备处理数据之前就向其发送数据，那么就会增加DMA缓冲区溢出的风险。此外，如果实时应用在FPGA发送数据之前就开始寻找数据，那么实时应用程序就可能发生超时。当实时应用程序准备好接受数据时，可以利用一个简单的中断来同步FPGA的数据采集与实时应用程序。

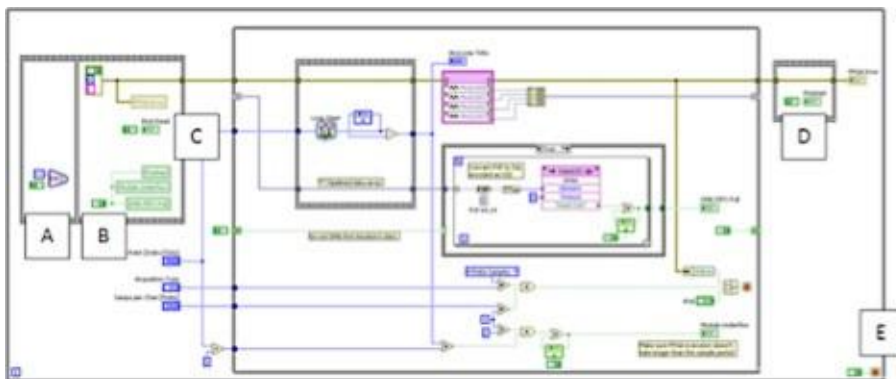


图6.22 与主机同步的DMA采集

接下来，将解释个例子中的代码是如何运行的：

- A. 在应用程序开始运行时，FPGA 仍然维持中断并等待主机应用程序识别这个中断。
- B. 一旦主机确认中断，错误显示器就立即初始化，采集循环开始运行。
- C. “第一次读取”指示器通知实时应用程序一个新的数据采集开始运行。实时应用程序将这个标识作为LabVIEW波形数据开始采集的时间戳。
- D. 如果采集数据集被错误状态或主应用程序终止，“完成”指示器就立即被设置以来确认采集循环已经终止。
- E. 外部while循环重新运行FPGA代码来进行另一个数据采集任务。在处理新的采集任务之前，它仍然维持中断，直到被主机应用程序识别。

板载标定与通道总数确认

数据被放入DMA FIFO时，所有通道交织在一起。读取数据时，实时应用程序需要重新整理这些数据并将它们按照一定的次序编织成二维数组。如果具有四个通道，那么实时应用程序必须将一维FIFO数组重组为一个具有4列的2维数组。因此，在数据采集集中，就有必要使实时应用程序与FPGA VI的通道数量保持一致。

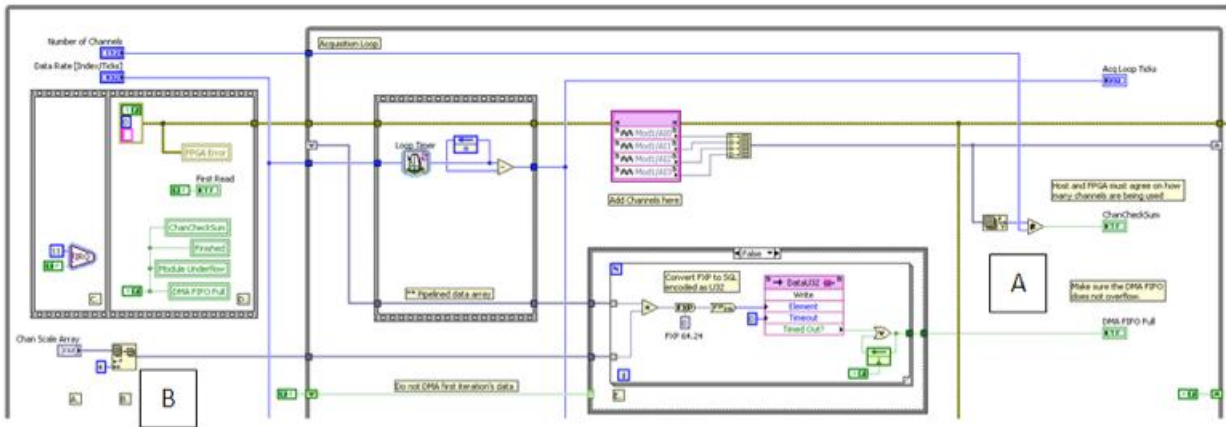


图6.23通道总数检查和板载标定

接下来，将解释个例子中的代码是如何运行的：

- A. 检查实时应用程序中的预期通道数，并将其与FPGA上的实际通道数进行比较。如果通道数量不吻合，则“ChanCheckSum”标识就会报告这些差异。这就是通道总数检查。
- B. 除了检查正确的通道数外，上面的例子也为每个数据点进行了标定。简单地将初始电压乘以一个标定值，就可以将加速度计、麦克风、甚至一些应变片标定为正确的工程单位。“通道标定数列”会从实时应用程序的每个通道接收一个标定值。在for循环中为每个通道的初始电压乘以一个相应的标定值就可以进行标定。现在从FPGA上返回的数据就以工程单位为其单位，而不是V或mV/V。如果需要，还可以增加更加复杂的标定。

使用实时应用程序读取DMA FIFO

实时应用程序需要能设定采样率和采样模式，配置实时DMA缓冲区，识别FPGA的“开始”中断，访问DMA缓冲区的样本，读取DMA缓冲区的数据，重组交错数据，以及将数据转化为可以使用的形式。

可以利用FPGA Interface VI人工地实现这些功能。但还可以使用一组专门设计的VI来进行DMA波形数据采集。这些VI内包含在开发人员指南中。通过访问ni.com，可以得到最新的VI以及更详细的说明。

Reference Applications for CompactRIO Waveform Acquisition

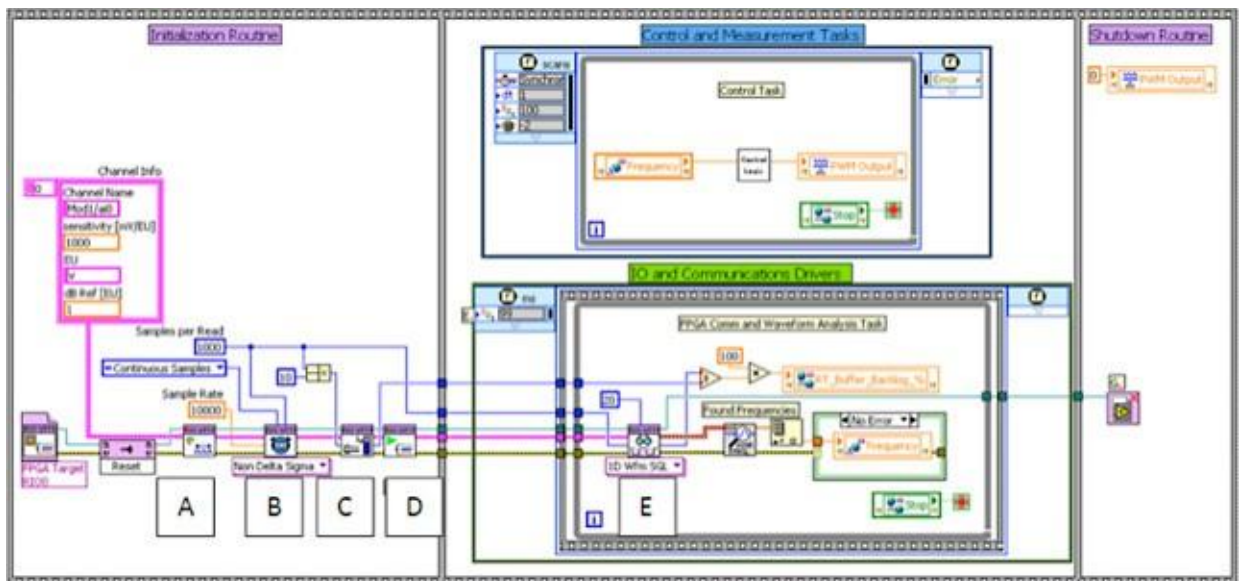


图6.24使用CompactRIO波形数据采集VI

CompactRIO Waveform VI完全是用LabVIEW代码编制的，并且使用的FPGA Interface VI与之前讨论的例子相同。接下来，将解释个例子中的代码是如何运行的：

- A. 该VI收集标定和通道名称等通道信息。通道信息（Channel Info）控制中的元素数目必须与FPGA获得的通道数相等。标定值和通道数被传送到FPGA上。
- B. 定时 VI设置采样率和采样模式，并将这些信息传送到FPGA。此外，如果采样模式设为“有限”，VI将自动配置DMA缓冲区，使其完全等于采集所需的样本数。因为这个应用程序使用“连续”采集，所以缓冲区需要更多的手动控制。
- C. 使用连续采集时，缓冲区的大小必须数倍于所要读取的数据。如果实时应用程序不能快速从DMA FIFO中读取数据，那么数据就会占满DMA缓冲区会并使其发生溢出。为了避免这种情况，必须降低采样率，提高采样数量，或者在采集过程中减少运行在实时应用程序上的进程数。有时候，需要综合考虑这些因素，才能为应用程序和控制器找到一个合适的方法。
- D. Start VI运行已编译的位文件，等待PFGA中断，之后对中断进行识别，然后进行数据采集。
- E. Read VI轮询实时DMA缓冲区，并从DMA缓冲区中读取采样数据。另外，Read VI还要检查运行FPGA上的所有错误条件（缓冲器溢出、模块下溢、读取超时等等）。Read VI也可以采取很灵活的方式将数据传递给应用程序；可以以一维波形数组、二维单精度浮点型数组、或一维U32数组的形式返回数据。一定要记住，U32数值实际上是将单精度浮点值作为U32编译而成的。之后需要一个简单的类型转换将所有的数据转换为正确的格式。

实时应用程序连续读取四个通道的数据，将其转换为一维LabVIEW波形数组，并为每个通道确定最高采样频率，最后将0通道的频率传递到存储表。这样就可以在平行的控制循环中使用这些数据。

使用Delta-Sigma-Based C系列模块采集波形数据

并不是所有的模拟输入C系列模块都使用相同的模数转换（ADC）技术。一些模拟输入模块使用逐次逼近（SAR）ADC，一些使用户增量累加（delta-sigma）ADC。本文将用FPGA实例来说明如何从逐次逼近（SAR）或非增量累加（non-delta-sigma-based）C系列模块中采集数据。如果使用的是NI 9234、NI9237或其它增量累加（delta-sigma-based）C系列模块，就必须对FPGA进行特别考虑。

逐次逼近设备与增量累加设备的主要区别是如何在FPGA上对采集循环进行定时。对于SAR模块来说，循环计时器控制着循环速率，为主应用程序提供所需的采样频率。必须手工进行下溢条件检查。

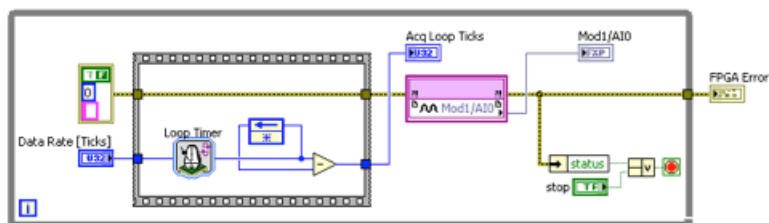


图6.25 对使用逐次逼近C系列模块的FPGA采集循环进行定时

图6.26所展示的程序是从增量累加C系列模块中采集数据的。注意程序中是怎样设定采样速率的以及计时是从采集循环外开始的。在这个程序中，FPGA I/O Node根据Data Rate 枚举中的数据来控制循环速率。调用FPGA I/O Node之后，它将执行等待，直到从模块中传来一个有效的速率。如果任何一个循环所执行的时间超过了设定的采样周期，就会产生下溢并且I/O Node会发出警告65539。增加内处理或者DMA FIFO时，一个很好的方法就是使用数据流来给FPGA I/O Nodes足够的时间从模块中检索数据。

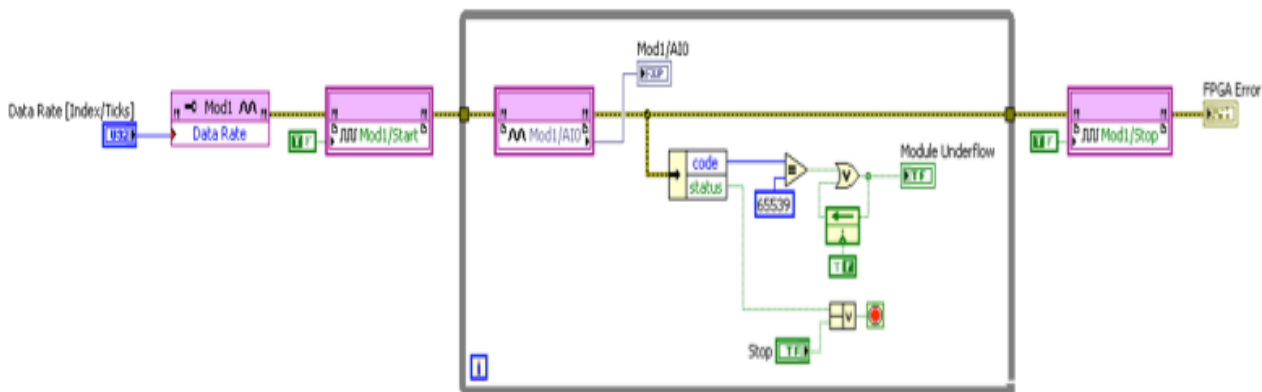


图6.26对使用增量累加C系列模块的FPGA采集循环进行定时

FPGA VI也应包含数据流操作、溢出检查、一个DMA FIFO以及数据变换。可以在开发者论坛(NI Developer Zone)中查到关于增量累加

模块的更详细的列子

Reference Applications for CompactRIO Waveform Acquisition

不含扫描模式的C系列模块

大多数C系列模块都支持扫描模式，但一些特殊的模块，比如电机驱动器、CAN、串行接口、SD卡模块，就不支持扫描模式。另外，CompactRIO是开放的系统，所以用户和第三方可以创建C系列模块。要想在CompactRIO系统中使用这些模块，就必须确保将这些模块处在项目中的FPGA下，这样就可以将他们放置在混合模式里。因为这些模块不是典型的模拟或数字模块，所以没有通用的API，但NI模块在NI Example Finder 里提供了一些实例示例，包含了LabVIEW FPGA和LabVIEW 实时主机代码。同时，大多数第三方模块也装载了实例。

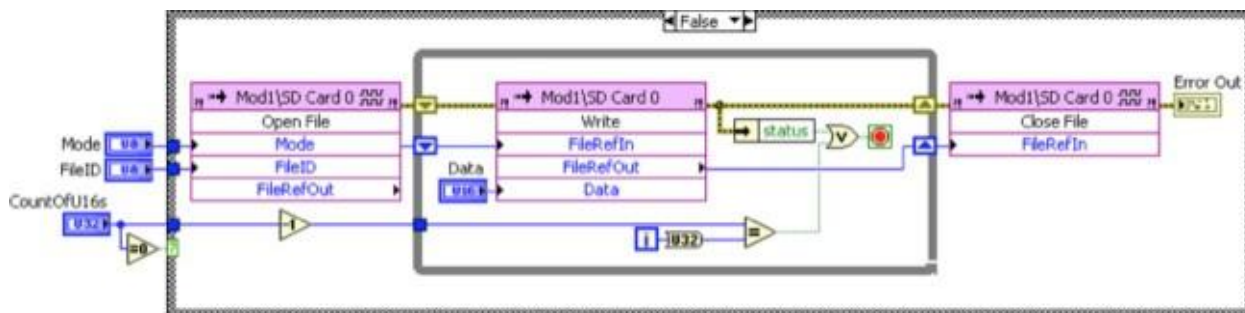


图2.67向NI9802 SD卡写入数据的例子

LabVIEW FPGA开发的最佳技巧

本章将涵盖许多高级技巧。当使用LabVIEW FPGA和CompactRIO创建高性能控制系统的时候，使用这些技巧可以缩短开发时间。这些技巧包含调试技术，比如仿真以及一些推荐的编程技巧、常见错误的避免方法、创建快速、有效、可靠的LabVIEW FPGA应用程序的多种方法等。

必须具有LabVIEW FPGA编程技术的基本知识，才能从这节中学到更多的东西。

本节将提供一个为有刷直流电动机创建高性能控制系统的例子。将探究在LabVIEW FPGA子VI中使用的一系列编程技术，这些技术用于生成PWM驱动信号、编译正交编码传感器传来的数字脉冲信号以及执行PID控制来关闭电机位置循环。最终会创建一个高性能的控制系统。这个系统将拥有亚毫秒的定时抖动和多个40MHz的处理循环，而它仅占3M gate FPGA的17%。

接下来将研究五个关键的开发技术，这些技术用来创建可靠的高性能LabVIEW FPGA应用程序。

技巧1. 使用单周期的定时循环（SCTL）

第一个开发技巧是在LabVIEW FPGA中使用单周期的定时循环，或SCTL。

使用SCTL向程序代码添加特定的定时限制，使程序在FPGA时钟的一个单位时间内执行，这样就能使LabVIEW FPGA编译器从代码内部进行优化。在SCTL内进行的代码编译将比在普通的while循环内进行的相同编译占用更小的FPGA空间。而且SCTL内代码的运行速度是非常快的。在默认的40MHz时钟速率下，每个循环的运行时间仅相当于25ns。

图6.28展示了两个相同的LabVIEW FPGA 应用程序，左边的程序使用了标准while循环，右边的程序在子VI中使用了SCTL。这个例子阐明了并行处理的强大功能。上边的循环读从发动机的正交译码传感器上读取数字信号并进行处理，下边的循环执PWM来控制向发动机输送的功率。这个应用程序是为NI9505发动机驱动模块编写的，用来控制有刷直流发动机。这些代码运行的速度非常快，右面的应用程序以40MHz的时钟速率运行两个不同的循环。

While循环

SCTL

使用的门数量：3245，占22% 使用的门数量：2456，占17%

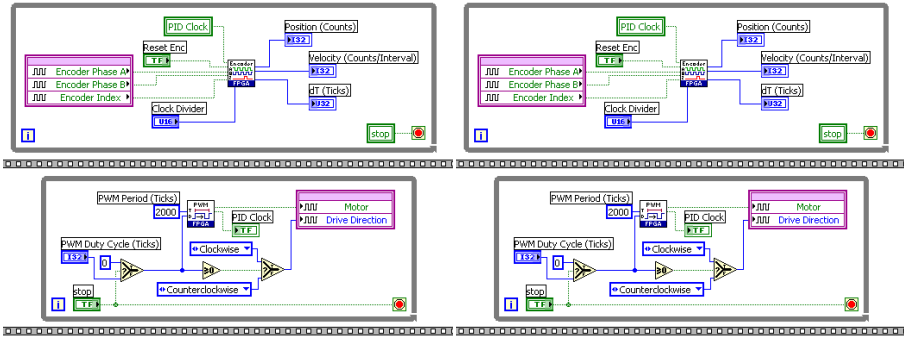


图6.28平行处理的功能

同时也显示了编译报告的结果。使用SCTL建立的应用程序占用更少的SLICE，但它要花更长时间来编译，因为编译器必须满足SCTL的时间限制。

现在更深入地研究一下SCTL是如何工作的。

当在标准while循环中对代码进行编译时，LabVIEW FPGA顺序地给每个函数的时钟数据插入触发器，从而加强LabVIEW的同步数据流性质并防止出现竞争状态。图6.29中，在每个函数的输出端使用FFs框标记了触发器。

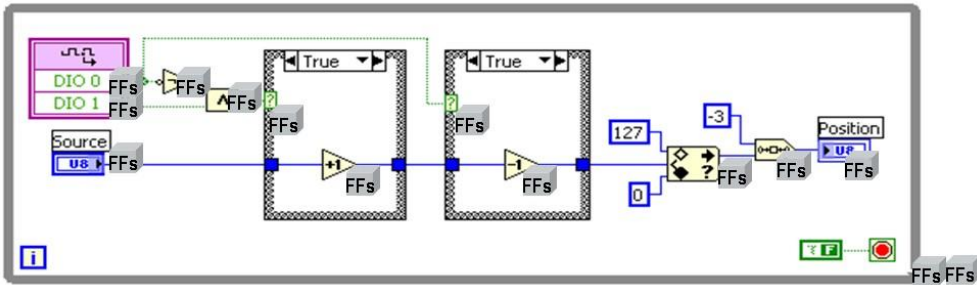


图6.29当在标准while循环中对代码进行编译时，LabVIEW FPGA顺序地给每个函数的时钟数据插入触发器

图6.30展示了相同的代码在SCTL中的编译过程。只有循环的输入和输出端拥有触发器。它以一个更加合理的并行方式，以及更加精简的逻辑运算来执行内部代码，从而优化输入与输出之间的代码。

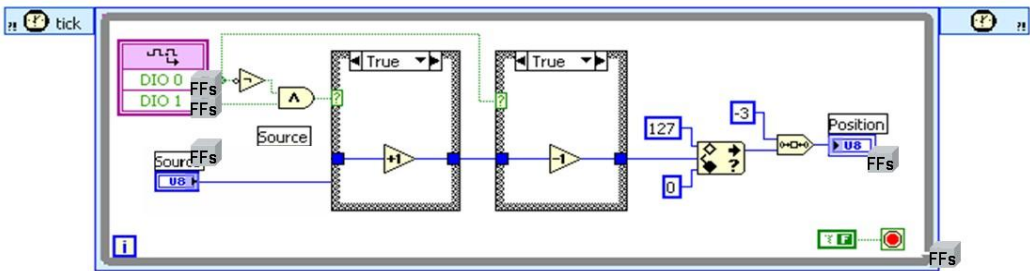


图6.30将图6.19中的代码编译到SCTL中


正如上面所描述的那样，SCTL是一个优化LabVIEW FPGA代码的简单途径，但一些它也存在局限性。其局限性如表6.1所示：

SCTL不支持的项目	建议使用以下的选择
长序列的串行代码	使用更加并行的方式来运行代码。向数据线插入反馈节点来增加数据流
求商和余数	使用Scale by Power2进行整除，或者使用定点数学库
循环计时器和等待函数	使用Tick Count函数来触发事件
模拟输入和模拟输出	放置在一个单独的while循环中，并使用局部变

	量来发送数据
While循环	对于嵌套的子VI，使用反馈节点来保持状态

表6.1SCTL的局限性

为了使用SCTL，循环内的所有操作都必须与FPGA时钟的一个周期相协调。在编译进程的开始阶段，如果SCTL没能为编译器生成正确代码，那代码生成器就会产生错误信息。这就意味着长序列的串行代码可能不适用于SCTL。所谓串行代码就是下一步操作必须使用上一步运算结果的代码，这样就不会出现并行运算。为了解决这个问题，可以重新写入数值，使程序更加并行。比如，可以插入一个反

馈节点（），将上一步的计算结果传递至下一个循环中以供运算，这就是所谓的数据流。使用SCTL，在多个循环迭代中将程序代码分解成多个片段，这样使用数据流技术就能减少每次运行的时间。

SCTL不支持使用Quotient and Remainder函数（求商与余数的函数）。如果需要对数据做整除运算，就可以使用Scale by Power of 2函数。使用这个函数，可以乘以或除以2的次幂，即2、4、8、32等等。对于一个定点结果，可以使用LabVIEW FPGA 的Fixed-Point Math Library（定点数学库）。图6.31所示为定点除法子VI以及配置面板，它包含了Execution Mode控制。Execution Mode控制能使这个子VI在SCTL内使用。

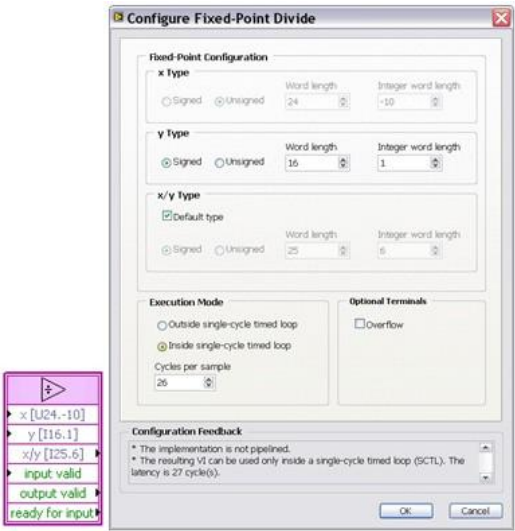


图6.31定点除法子VI以及配置面板

Fixed-Point Math Library包含LabVIEW FPGA IP模块，这些模块能够执行多种初等数学函数和超越函数。这些函数使用LabVIEW8.5中介绍的定点数据类型，它提供了更过的扩展功能，包括除法、正弦、余弦。所有这些函数都可以在开发计算机的SCTL以及Windows和FPGA仿真中使用。Library提供了每个函数的信息帮助文件。可以从ni.com免费下载到。

如果想要创建一个子VI来在SCTL内部执行，可以使用反馈节点来保存子VI里的状态信息。这样就能避免在SCTL中使用while循环。图6.32展示的LabVIEW FPGA例子使用Fixed-Point Math Library中的函数，来计算直流发动机的一个微分方程。在每个定点数学函数之后，都有一个反馈节点来保存计算结果，并将结果迭代传递到下一个循环中。然后用带有反馈节点的Tick Count函数（在右上角）来计算子VI的循环速率。

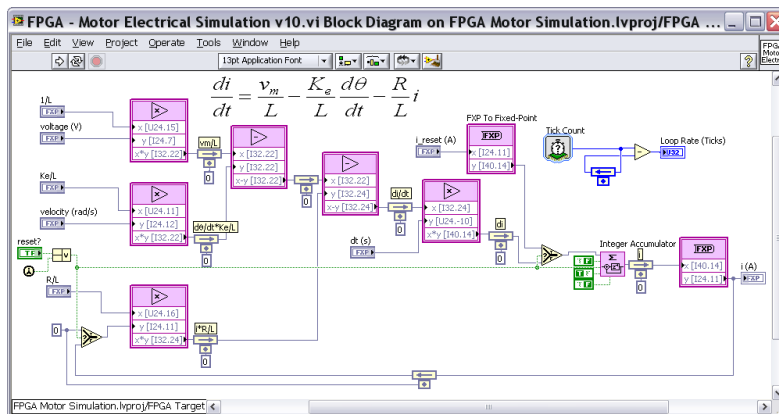


图6.32使用Fixed-Point Math Library中的函数，来计算直流发动机的一个微分方程

如图6.33所示，FPGA应用程序中的顶层SCTL，调用电机模拟子VI。

因为子VI是嵌套在SCTL内的，所以**Loop Rate (Ticks)**的返回值始终为1。然而，数据流会在子VI的电压输入到电流输出之间产生6ms的时间延迟。

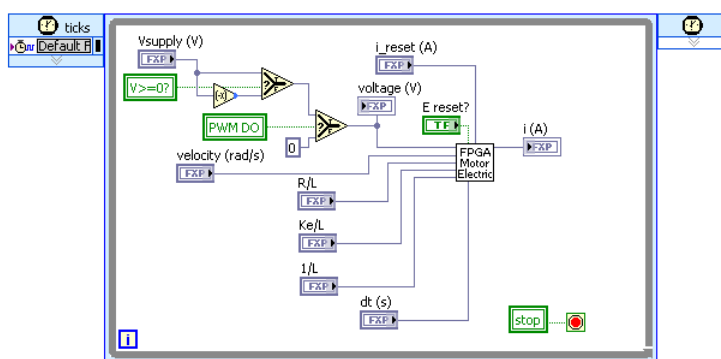


图6.33数据流在子VI的电压输入到电流输出之间产生6ms的时间延迟

除了数据流，也可以使用**SCTL**内的状态机通过一系列有序的状态来更好地组织与运行代码。状态机的基本结构是一个选择结构，在结构里包含了下一个循环要执行的状态，并通过移位寄存器将这个状态传递到下一个循环中。如果子VI被放置在一个**SCTL**中，那么每个状态就必须能在一个时钟周期内完成运算。另外，还可以使用移位寄存器和一个计数值来实现**for**循环的功能，或为程序的执行添加一个特定的等待状态。

注意：如果给程序添加循环计时器或等待函数，就会使程序的运行时间大于**1ms**，这样就不能在**SCTL**中运行这个程序。模拟输入和模拟输出函数的运行时间要远大于**1ms**，所以不能在**SCTL**中使用。但是，可以将它们放入一个标准的**while**循环中，并使用局部变量来与**SCTL**共享数据。

贴士：创建计数器与计时器

如果需要在程序运行一段时间后触发一个事件，就可以使用**Tick Count**函数来计算程序已经用掉的时间，如图6.34所示。一定不要使用**while**循环和**SCTLs**自带的循环端子，因为它会在最大值处达到饱和并停止运算。经过**2,147,483,647**次循环后循环端子就会饱和。如果以**40MHz**的时钟速率运行，那么仅需**53.687**秒，它就会达到饱和。可以使用一个无符号整型数据和一个反馈节点以及**Tick Count**函数来创建基于**40MHz** FPGA时钟的定时器。

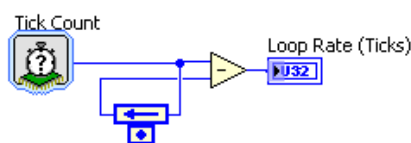


图6.34使用Tick Count函数来计算已经用掉的时间

因为使用无符号整型数据作为计数器的数据，所以当计数器进行反向运算时，得到的程序已用掉时间结果仍然是正确的。这是因为如

果从一个计数值上减去一个无符号整型数值，即使计数器发生了溢出，得到的答案也是正确的。

另一个常见的计数器类型是循环计数器，它运来计算一个循环已经执行的次数。使用循环计数器时，通常选取无符号整型作为其数据类型，因为它能提供一个最大的数值范围。无符号64位整型数据提供的巨大计数范围大约为1800亿亿（18billion-billion）。即使FPGA时钟以40MHz的速率运行14,000多年，计数器也不会发生溢出。

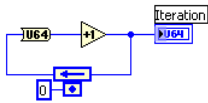


图6.35使用循环计数器时，通常选取无符号整型作为其数据类型，因为它能提供一个最大的数值范围
接下来将学习另一种编程技术，来编写优秀而有效的LabVIEW FPGA代码。

技巧2. 使用模块化、可重用的子VI来编写FPGA程序代码。

接下来将学习模块化编程技巧——将整个程序分成几个相互独立的程序块，可以单独对这些程序快进行设置以及测试等。这看起来像是一个简单的概念，但对FPGA的开发来说，这样做会给程序的开发带来很多方便。比如，设计一个用来计算循环速率的函数。在循环内放置Tick Count函数，用来读取当前FPGA时钟，并减去之前存储在移位寄存器中的时间值。此外，还拥有一个64位计数器，每次调用这个函数时，计数器都会增加一个数值。这个函数使用了SCTL，所以函数运行一次只需要25ns的时间。所以将这个子VI放置在一般的while循环内，不会影响循环运行速度。

前面板如图6.36所示。子VI右边的两个端子连接两个指示器，这样数据就可以传送到顶层LabVIEW FPGA应用程序里。

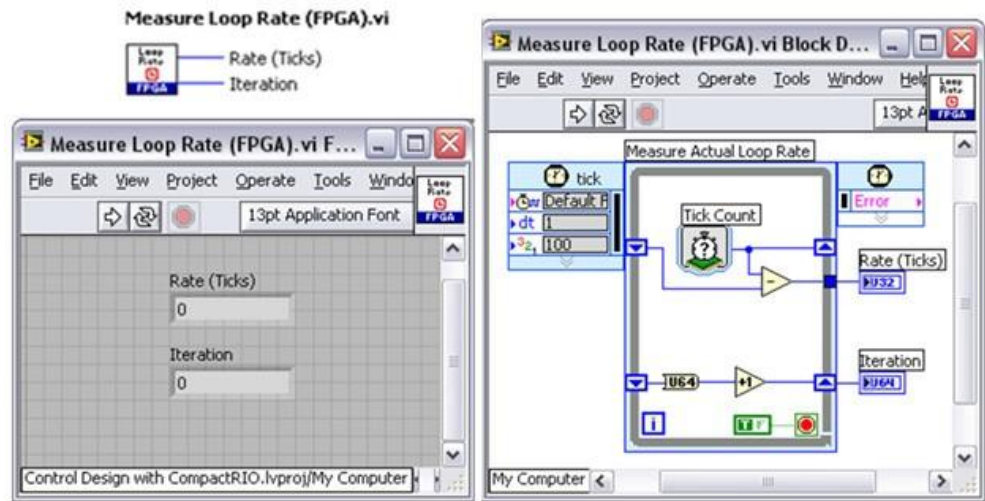


图6.27计算循环速率的子VI的前面板，将这个子VI放置在一般的while循环内，不会影响循环运行速度
这个函数在图6.27的程序中使用过。将此子VI放置在另一个循环内来计算顶层程序的运行速率
回顾一下这种编写代码方式的最大有点，见表6.2。

优点	说明
更易于调试和错误检查	可在编译前，在Windows上调试代码
更易于记录和查看变化	可以在VI说明中添加帮助信息
创建更简洁、易懂的顶层程序	与其他编程相比，代码更直观

表6.2 使用模块化、可重用的子VI来编写FPGA程序代码的好处

编写模块化代码通常是一个很好主意。然而当设计FPGA逻辑时，会发现这样做还有更多的好处。
第一，代码更易于调试和错误检查。一个很大的优点就是：在编译子VI之前，可以先在Windows上调试这个子VI。可以参考本节稍后的列子。

第二，更易于记录和查看变化。这是因为代码是模块化的，并且可以在VI说明中添加帮助信息。

第三，可以在子VI中预设一些功能，这样程序使用子VI时，就更加简洁、易懂，且子VI具有更好的可重用性。将提供给程序员的功能选择性设置为子VI的端子。大多数时候，用户不需要修改底层的代码，只需为这些子VI提供一些参数，比如例子中的**Pulse Width Modulation (FPGA). Vi**。

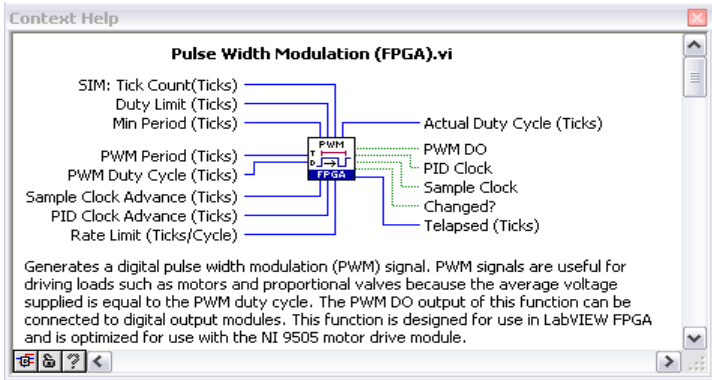


图6.38值需给予VI提供一些参数

接下来将学习创建模块化、可再用子VI需要注意的一些事项。

贴士：将逻辑与I/O分离

首先，将I/O节点放置子VI之外。这样程序就更加模块化，并使顶层程序代码更加易读。特别是对于控制应用程序，一定要确保所有顶层I/O选择性清晰可见。如图6.39所示：为NI9505电机驱动模块所写的PWM循环。

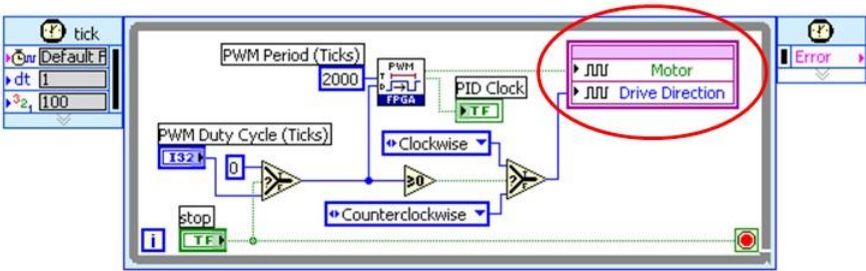


图6.39将I/O节点放置于子VI外，并且一定要确保顶层程序的所有I/O都清晰可见

使用一个端子将数据从子VI传递到顶层程序里，而不用在子VI中嵌入I/O节点。这就使得FPGA程序更容易调试，因为可以使用仿真I/O在Windows中测试这些子VI。

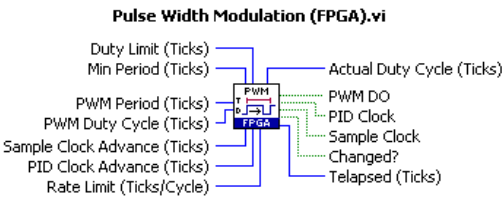


图6.40使用一个端子将数据从子VI传递到顶层程序里，而不用在子VI中嵌入I/O节点

采用这个方法也可以减少附加的I/O节点，否则这些节点被子VI多次调用,会导致逻辑门的浪费。这是因为子VI调用这些节点时，会编译这些节点来增加额外的逻辑运算来处理多个调用者对相同共享资源的访问，

最后，这个方法使高级应用程序具有更强的可读性——所有I/O读写操作都明确的显示在程序框图上，没有隐藏起来。

当使用上面的方法来编写函数块时，子VI需要有一定的逻辑存储能力来保存状态值，比如已经使用的时间，并将此信息从这个循环传递到下个循环。

贴士：在函数块中存储状态值

图6.41显示了如何在循环中添加移位寄存器来将信息从一个循环传递至下一个循环。每次调用这个函数时，循环计数器就会增加一个单位值。

注意将一个常数连线到循环条件端子上，这样每次调用函数时循环只运行一次。在这个程序里，并没有真正的使用循环——而是简单地利用SCTL来优化代码并用移位寄存器来存储状态值。

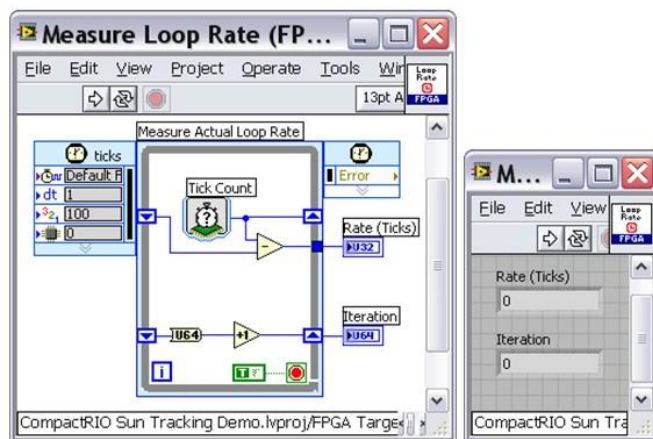


图6.41给循环增加移位寄存器，将信息从一个循环传递到下一个循环

注意：必须初始化子VI的移位寄存器来保存状态。在第一次调用时，移位寄存器的值是默认的数值类型的值，对于整数类型，默认值为0；对于布尔类型，默认值为False。如果需要将移位寄存器初始化为其它数值，可以使用**First Call?**函数和**Select**函数。

读者可能会问如何在SCTL内创建模块化得函数块，因为SCTL不允许嵌套另一个SCTL。

如图6.42所示，可以使用反馈节点来完成这个任务。此方的法主要优点是可以很容易地初始化反馈节点，并将子VI插入一个高级SCTL，这是因为它不包含嵌套循环结构。

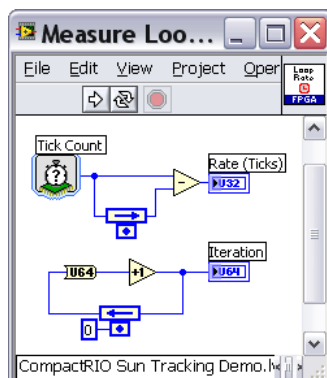


图6.42 使用反馈节点在SCTL内创建一个模块化的函数块

第三种方法是使用**VI-scoped**存储器。下图是这个储存器的程序图，可以在局部使用此子**VI**并不需要将其添加到项目里。这使得子**VI**具有更强的模块化以及在项目之间移动子**VI**时更加方便。

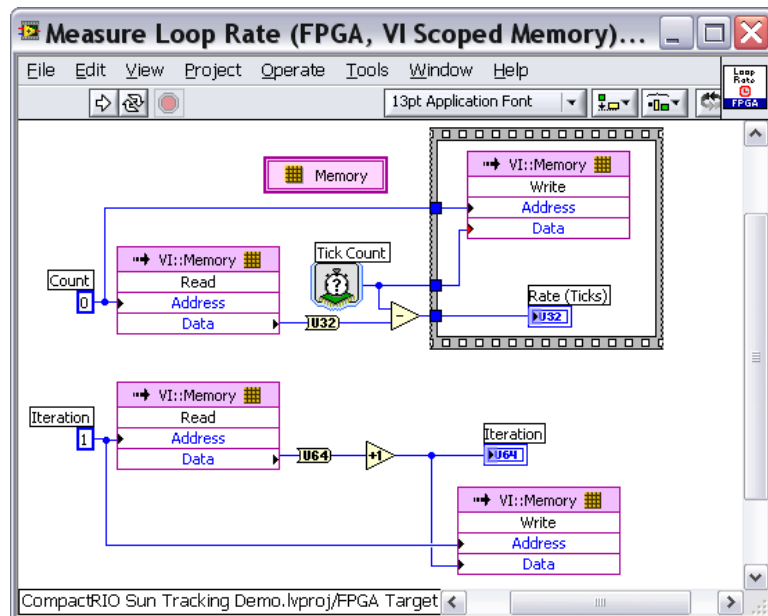


图6.43使用带有VI-scoped存储器的子VI，使得程序更加模块化

在这个简单的例子中，对应用程序来说，使用VI-scoped存储器好像有点浪费。因为程序拥有两个存储地址而每个存储地址只存储一个数据点。但是对于需要储存数列的应用程序来说，VI-scoped存储器是个功能强大的工具。一般来说必须避免使用一个很大的数列来储存数据，而应该使用VI-scoped存储器。

实时更新查询表（LUT）

通常使用控制应用程序的本地存储器来存储列表数据，比如非线性传感器的校准表、数学方程（比如对数或指数）或是可以通过表格地址索引的任意波形。图6.44显示了基于FPGA的查询表（LUT），表中储存了10,000定点值并在数值点之间执行线性插值。因为使用了VI-scoped存储器，所以在应用程序运行期间也可以改变LUT值，而不需要重新编译FPGA。

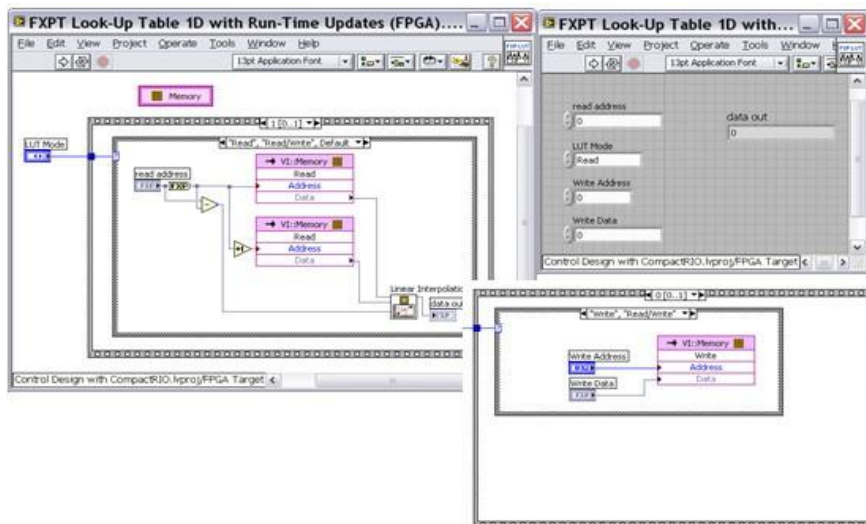


图6.44使用基于FPGA的LUT来储存10,000定点值并在数值点之间执行线性插值

在这个例子中将学习VI-Scoped 存储块的配置。可以配置存储器的容量和数据类型，甚至可以定义存储元素的初始值。

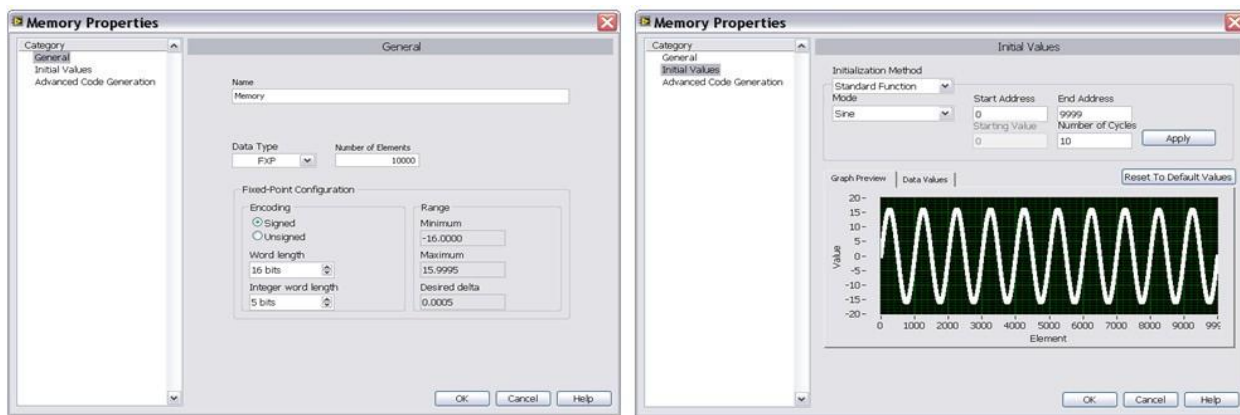


图6.45本例中VI-Scoped储存器的属性配置页

接下来将学习创建模块化的FPGA子VI的另一个小提示。

贴士：不要在子VI中放置延迟计时器

通常避免在模块化子VI中使用循环计时器或等待函数。如果子VI没有延迟，那么它就会以最快的速度运行，这样就会使调用VI的时间属性成为其内在属性。而且如果程序内部没有延迟函数，那么就可以更加容易地适应SCTL

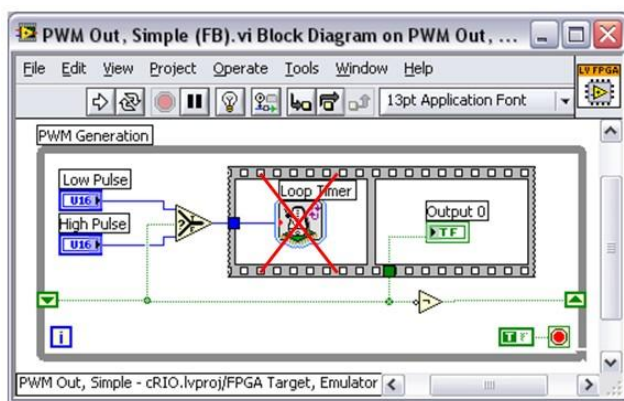


图6.46通常避免在模块化子VI中使用循环计时器和等待函数

图6.47左边的程序显示了如何改编PWM代码，使其使用Tick Count函数来代替Loop Timer函数。使用反馈节点来存储已运行的时间。在程序运行的过程中需要反复开关输出指示器，并在PWM循环结束时，需要重设已运行时间计数器。代码看起来可能有点复杂，但当将其放入顶层循环时，它不会影响循环的整体运行时间——这样移植起来就更加方便。

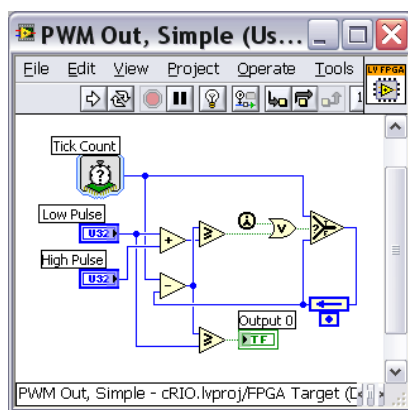


图6.47改编PWM代码，使其使用Tick Count函数来代替Loop Timer函数

在学习下一个技巧前，先来学习另一个需要注意的提示——配置程序代码，使子程序可以在同一个应用程序的多个位置出现，并且每个子程序的副本之间互相独立。

贴士：充分利用VI的可重入性

可重入执行是VI的一个属性。当设置VI为可重入执行时，就可以在程序的不同地方同时调用这个VI，并且为每个VI副本分配一个独立数据储存空间。

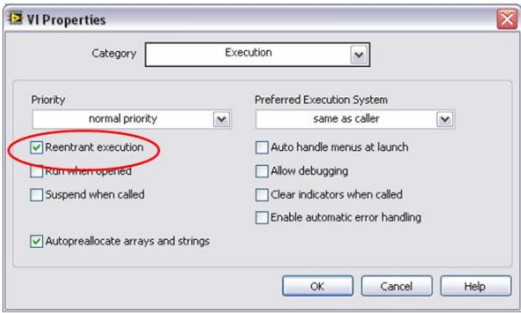


图4.68 配置VI为可重入执行

图6.49举例进行说明。在本例中，子VI被设为可重入执行，这样四个循环就可以同时运行，并且每一个循环内部的移位寄存器、局部变量或VI-scoped存储器数据相互都是独立的。

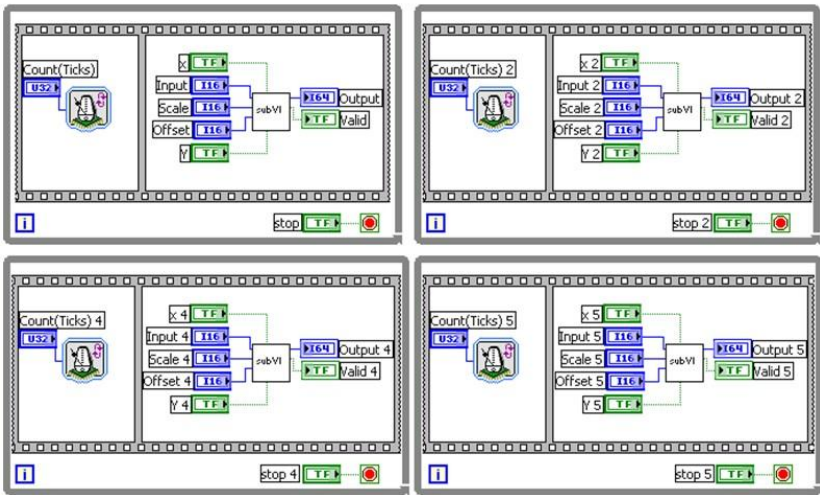


图6.49子VI被设置成可重入执行，这样四个循环就可以同时运行

在LabVIEW FPGA实例中，这也意味着每个函数副本都使用各自的PFGA内存。所以重入执行对于程序的移植性来说非常有用，但是这样会占用更多的逻辑门。

如果FPGA逻辑门资源短缺，可以将函数配置为多元化来代替可重入执行。本文就不在此研究这个高级技巧，只是进行概括的介绍。这个技术主要使用局部内存来储存每个调用循环的注册值，并使用一个整型的“ID 标识”来 辨认这些注册值。因为所有的循环都使用相同的FPGA内存（数据的储存地址不同），所以每个调用数据的程序都会阻止其它程序调用数据，这样就会导致程序的运行变慢。然而，如果重新使用相同的硬件SLICE，就会减少逻辑门的使用。对于那些FPGA比I/O快许多的控制应用程序，使用这个技术来节约逻辑门无疑是一个很好的选择。LabVIEW FPGA函数板上的一些函数使用多路复用技术来，能够实现使用最少的逻辑门进行高通通道计数操作。这些操作包括PID, Butterworth Filter（巴特沃斯滤波器）， Notch Filter（陷波滤波器）， 以及Rational Resampler函数。为了了解这些函数时怎么运行的，可以将其中的一个函数拖放到程序框图上并将其配置为多通道使用。然后右击函数，并选择Convert to SubVI来显示隐藏的代码。

现在可以回顾一下在上面一节中学习到的编写LabVIEW FPGA的高级技术。

技巧3：编译前使用仿真技术

这个技术的功能是非常强大的，使用这个技术不仅可以避免过长的编译时间，还可以获得更多的关于调试LabVIEW FPGA的方法。对于嵌入式系统的开发者来说，LabVIEW程序代码最重要的一个有点就是要具有可移植性。为LabVIEW FPGA编写的代码仍然只是LabVIEW 代码——可以在Windows上或其它设备和操作系统上运行。这些目标处理器的主要的区别在于代码的运行的速度和处理器是

否像FPGA一样支持真正的并行运算，或者像微处理器的多线程操作系统一样进行仿真并行运算。

LabVIEW FPGA能够以仿真模式运行整个LabVIEW FPGA应用程序，它与主机应用程序协同工作来进行代码测试。在测试过程中，可以让FPGA I/O读取随机数据，也可以自定义VI来生成模拟I/O信号。这对测试FPGA与主机之间的通信（包括DMA数据传送）特别有用。

然而，这种方法的不足之处在于整个FPGA应用程序都是模拟的。对于LabVIEW新函数的开发和测试来说，使用这中方法每次测试一个函数是非常有利的。本节主要研究函数仿真这个功能，它能使调试的程序“各个击破”。这样在将程序编译到FPGA之前，就可以分开测试每个函数。图6.50为两个运行在Windows上的函数仿真例子的前面板，这两个函数仿真用于测试和调试程序。

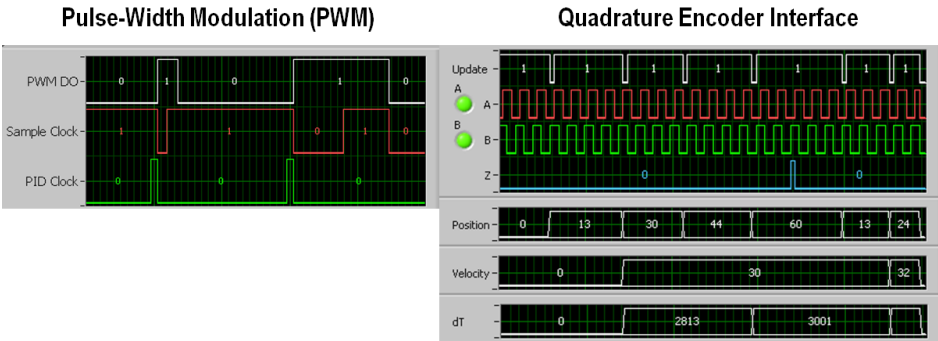


图6.50两个用于测试和调试的函数仿真的前面板

图6.51显示了用于调试PWM的LabVIEW FPGA 子VI的测试程序的前面板和程序框图。测试程序位于LabVIEW Project的My Computer里。当打开测试程序时，它就在Windows中开始运行。

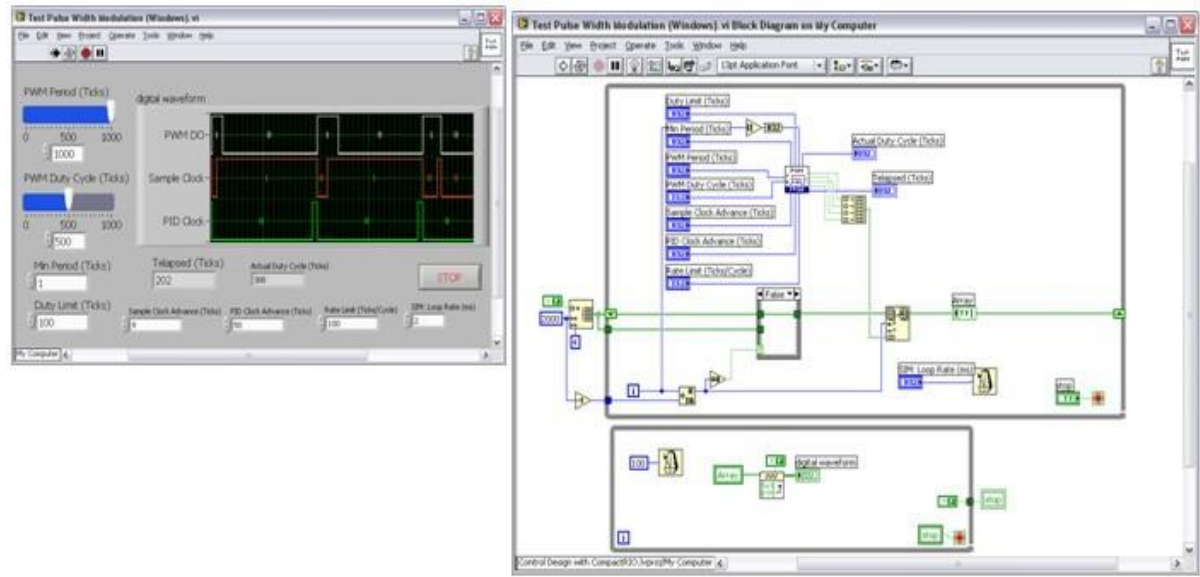


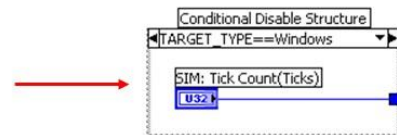
图6.51用于调试PWM的LabVIEW FPGA 子VI的测试程序的前面板和程序框图

贴士：为仿真提供Tick Count时间值

当为不同的处理器目标编译子VI时，使用LabVIEW的条件禁用结构来选择对哪一段代码进行编译。在这种情况下，当为FPGA编译代码时，就要运行一个Tick Count 函数，以及当代码在Windows上运行时，也要执行一个前面板控制。因此当在Windows中测试代码时，使用一个“仿真”时间计数值，来帮助创建位一准确和周期一准确模拟仿真。

这个技术被应用于图6.51的PWM测试示例中——当在Windows中运行子VI时，使用顶层while循环的迭代端子，将仿真FPGA时钟值传递至子VI。

Executes when code is compiled for Windows



Executes when code is compiled for FPGA

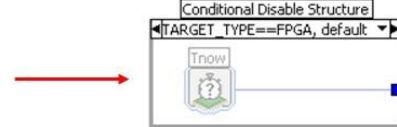


图6.52 当为FPGA编译代码时，运行一个Tick Count 函数，以及当代码在Windows上运行时，运行一个前面板控制

正如前文所述，可以使用函数仿真来测试程序。这样在编译之前，就能对FPGA逻辑有一个全面的了解。当运行仿真程序时，就可以使用所有的LabVIEW调试工具箱。可以创建“测试样式”来测试多种条件下的程序代码，否则这种测试将会非常复杂。在程序开发的过程中使用模拟仿真，可以完成以下功能：

- 快速循环以及添加功能
- 编译前，对LabVIEW FPGA有一个全面的了解
- 使用所有的LabVIEW 调试功能 (探针, 高亮执行等等)
- 在多种条件下测试程序代码

现在把仿真步骤再推进一步，在物理系统连接LabVIEW FPGA代码的情况下，可以创建一个可以精确模仿物理系统的动态闭环行为的仿真。

贴士：使用LabVIEW控制设计和仿真模块来测试LabVIEW FPGA代码

LabVIEW控制设计和仿真模块包含仿真机电系统的最先进技术就像用LabVIEW FPGA应用程序控制的直流电机。图6.53显示了有刷直流电动机的理论模型方程。直流电机由PWM斩波电路驱动并连接一个简单的粘性摩擦与惯性荷载。

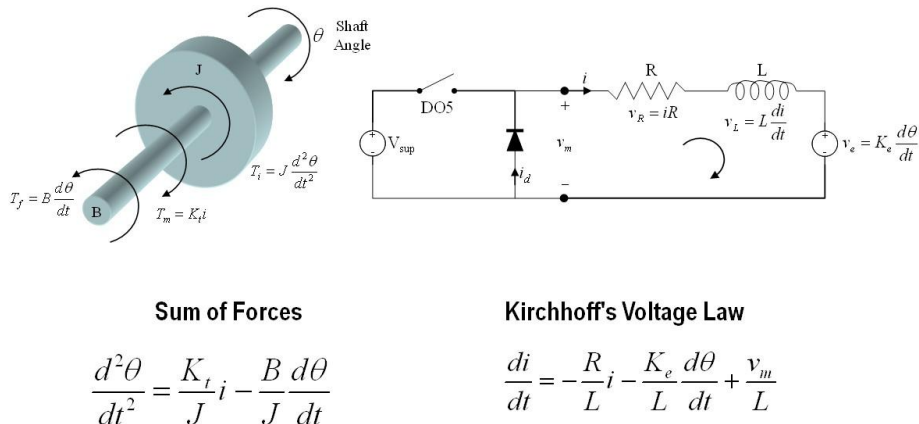


图6.53 有刷直流电机的模型及理论方程

利用LabVIEW控制设计和仿真子系统包括公式节点来控制有刷直流电机。将图6.53所示的两个微分方程以文本的格式输入到公式节点

中，如图6.54所示。然后利用积分函数（）从高阶倒数开始进行积分，比如从加速度积分得到速度，从速度积分得到位移。

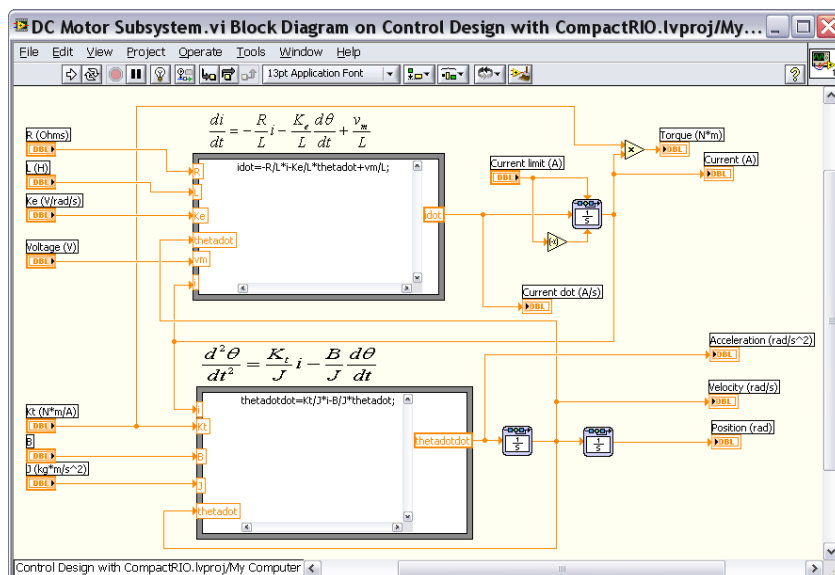


图6.54 图6.53中两个微分方程的文本格式

然后将Brushed DC Motor.vi 子系放置在顶层仿真循环内，并将其连接至LabVIEW FPGA函数，来模拟驱动电机的脉冲电压信号。最终就会得到一个非常逼真的闭合循环，用来模拟连接至实际的电机系统时，LabVIEW FPGA程序是如何运行的。

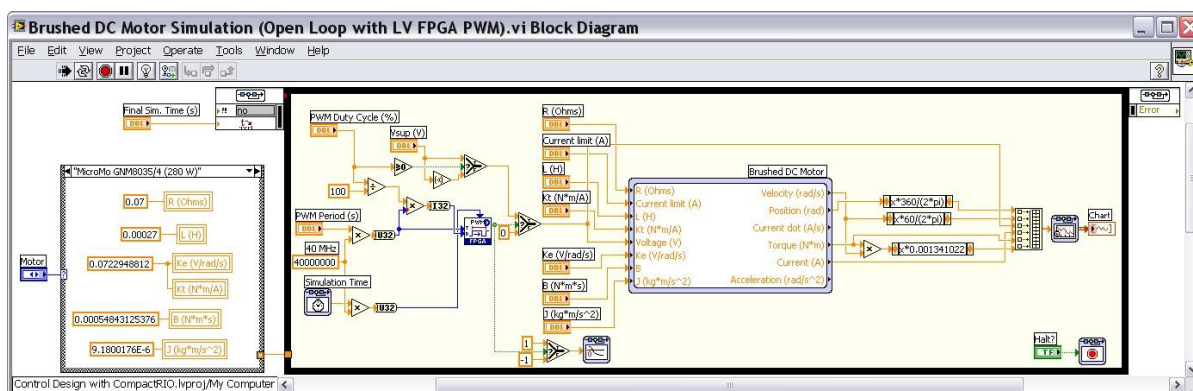
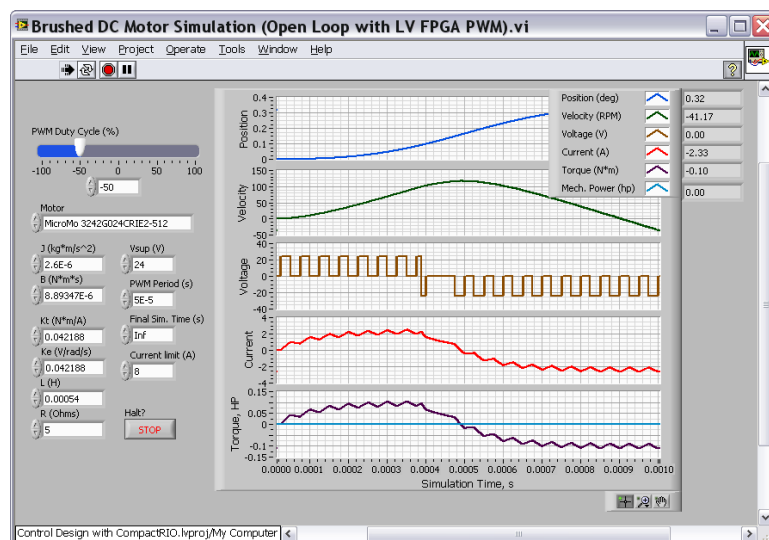


图6.55非常逼真的闭合循环，用来模拟连接至实际的电机系统时LabVIEW FPGA程序是如何运行的。

在本实例中配置LabVIEW FPGA应用程序，使用NI9505电机驱动模块控制电机。将模拟结果与实测结果进行比较，可以发现模拟波形和实测波形几乎完全相同。

使用函数仿真这个方法，除了可以测试程序外还可以得到关于程序的更多信息。将这个模拟仿真看成是一个虚拟机模拟器，可以用它来预测实际工作中代码是如何运行的。可以使用模拟仿真来制定设计决策、评估性能、选择组件以及检测最坏的情况。甚至可以在模拟中调整控制系统的PID控制循环，以及产看不同的电机里和不同的荷载情况下，这些调整是否能正常工作。模拟仿真也能帮助使

用户为系统选择正确的物理元件，比如选择能够满足功能需求的最好的发动机。

技巧4 同步循环

控制定时以及同步LabVIEW FPGA

对大多数控制应用程序来说，代码执行时的定时问题对系统的性能和可靠性都是非常重要的。使用LabVIEW FPGA，不仅能够通过程序的定时来完全控制系统，而且这个过程也是非常迅速的。不像处理器，FPGA可以运行真正的并行运行代码，而不是每次只能执行一条指令。这就使得编程更加简单了，因为不必担心如何设置优先权以及怎样在不同的任务间共享处理器时间。每个控制循环都像一个专门设计处理器一样，完全用来处理自己的任务。其结果就是使代码具有高可靠性和高性能。这样做带来的一个好处就是在高速率运行时，控制循环通常更稳定、程序更易于调谐以及对于扰动的感应更加灵敏。在这个电机控制例子中，拥有两个不同的时钟信号——一个样本时钟和一个PID时钟。这些信号是在应用程序中生成的布尔信号，用于同步循环。可以使用这些时钟信号的上升边沿或者下降边沿来触发。

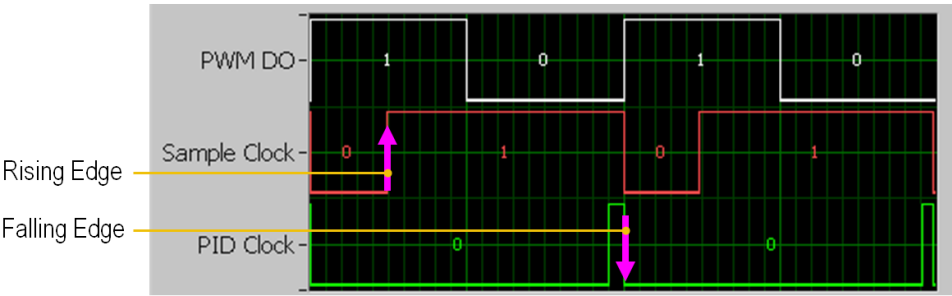
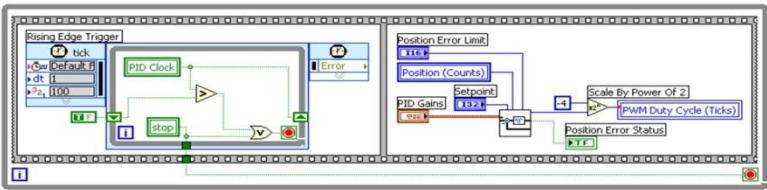


图6.56 拥有两个不同时钟信号的电机控制系统

接下来将学习用于检测这些信号的LabVIEW FPGA代码以及沿着上升边沿或下降边沿的触发。

通常情况根据布尔时钟信号来触发循环。首先等待出现上升边沿或下降边沿，当达到触发条件时，执行需要运行的LabVIEW FPGA代码。通常使用一个顺序结构来执行触发。如图6.57所示，使用顺序结构的第一帧来等待触发，使用第二帧来运行已触发的代码。

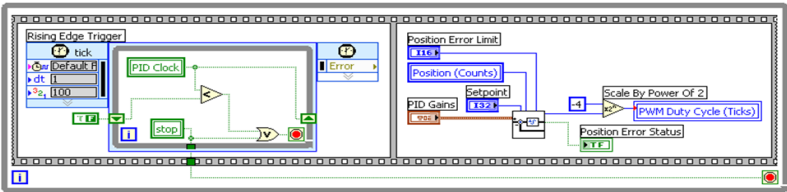
上升边沿触发：需要捕获到触发信号从假（或0）变为真（或1）的转变状态。通过将触发信号的值存储在移位寄存器中并使用Greater Than?函数进行比较，就可以得到触发信号的状态。（注：将True常量连线到移位寄存器的端子，对其进行初始化，避免在第一次循环时提早触发。）



Rising Edge Trigger

图6.57 上升边沿触发

下降边沿触发：使用Less Than?函数来捕获触发信号从真（或1）变为假（或0）的转变状态。（注：将False常量连线到移位寄存器的端子上，对其进行初始化。）



Falling Edge Trigger

图6.58 下降边沿触发

模拟电平触发：使用Greater Than?函数来捕获模拟信号是否大于模拟阈值的状态，并使用“Not Equal”函数输出的布尔值作为触发信号。这种触发实际上是捕获触发信号的上升或下降边沿，因为使用Not Equal?函数来捕获信号的任何一个变化。

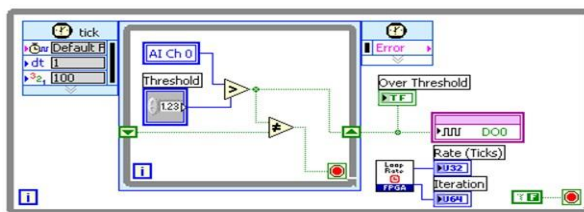


图6.59模拟电平触发

接下来将学习另一个常见的触发实例——触发事件发生时锁存触发信号值。

贴士：锁存触发信号值

在这个实例中，使用上升边沿触发，来将另一个循环中的**Analog Input**的值锁存在**Latched Analog Input**中。直到下一个触发事件发生，这个值将一直保存在**Latched Analog Input**中。但是，实际模拟输入操作是发生在另一个循环中的，使用局部变量在不同的循环间进行通讯。（注：使用局部变量在LabVIEW FPGA的异步循环间共享数据是一个很好的方法。）

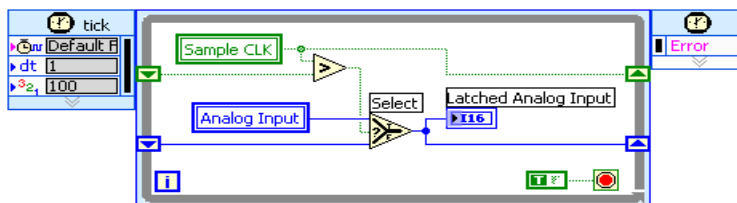


图6.60实际模拟输入操作发生在另一个循环中，使用局部变量在不同的循环间进行通讯

同步触发和锁存

图6.61所示的例子展示了这些触发与锁存技巧在实际中的应用。LabVIEW FPGA提供了真正的平行运算。在这个例子中，拥有三个独立的循环，就像三个专门设计的处理器在同一个芯片上同时运行。每个循环都只处理自己的任务，这样FPGA就拥有最高的可靠性。这也使得FPGA中编程控制的设计更加容易——不像使用一个处理器，不用担心添加新代码会降低程序的运行速度。

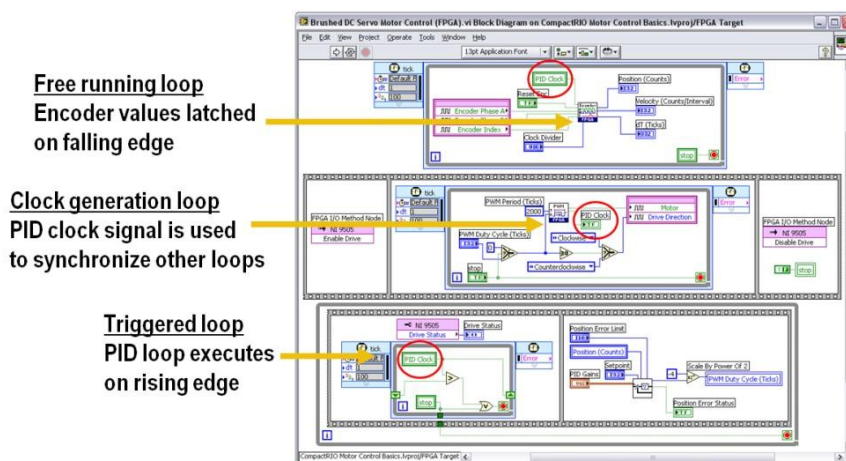


图6.61每个独立的循环只会处理自己的任务，这样程序就拥有最高的可靠度

- 其中一个循环用来生成同步时钟，以供其他循环使用
- 译码器函数需要以最快的速度运行，以避免漏掉任何数字脉冲。此函数以40MHz的速率运行，但使用其它循环来锁存位置（计数）信号和速度（计数/间隔）信号以同步数据。
- PID函数需要以一个特定的速率运行（20kHz或2000毫秒），并需要避免计时过程中的任何抖动。这是因为积分与微分步长取决于时间间隔，Ts。如果时间间隔不同，或同一个数值被多次传递到函数中，那么积分和微分步长就会错误。
- 在底部循环中，使用PID时钟信号的上升边沿来触发程序。读取SCTL中信号的局部变量，当捕获到上升边沿时退出循环。然后执行32位PID算法，此算法包含在LabVIEW的NI SoftMotion Development Module中。它读取命令地址，将其与编码

器测量的地址相比较，然后为PWM循环生成一条指令。在这个例子中，使用Scale by Power of 2函数将PID输出信号除以 2^{-4} ，相等于除以16。这就将时间缩放至±2,000毫秒的范围内，这也正是PWM函数需要的范围。1000 毫秒相当于半个工作周期，因为PWM的周期是2000 毫秒。

- 注意:顶层的两个循环以40MHz的速率运行。底部循环循环使用PWM时钟信号触发，以20kHz的循环速率运行。（触发时，NI SoftMotion PID函数需要36毫秒的运行时间。）

技巧5：避免“Gate Hogs”

到此为止，已经学习了4个开发LabVIEW FPGA代码的关键技巧。接下来将学习最后一个技巧：避免gate hogs。这些通常看起来无关紧要的代码却占用了很多的FPGA逻辑门（也就是所谓的分区）。以下列举了三个最常见的项目：

大数组或簇：使用前面板指示器或控件创建一个大的数组或簇，是一个最常见的编程错误，这样会占用大量FPGA逻辑门。如果不需要使用前面板指示器与主机处理器进行通信，就不要创建。如果需要传递12个以上的数组元素，就使用DMA作为传递数据的方式。任何时候都要尽量避免使用像Rotate 1D Array这样的数组操作函数。

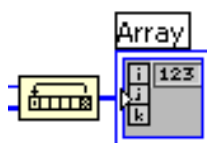


图6.62任何时候都要尽量避免使用像Rotate 1D Array这样的数组操作函数

商与余数：这个函数执行整除运算。输出的商是 $\text{floor}(x/y)$ ，即 x 除以 y ，舍掉商的小数位，所得的整数就是输出的商。输出的余数是 $x - y * \text{floor}(x/y)$ 所得到的值。比如23除以5，得到的商为4，余数为3。这个函数是门密集函数，并使用了用多个时钟周期来进行运算，所以不能在SCTL中使用它。使用这个函数时，一定要确保将所需的最小数据类型连线到端子上，尽量使用常量代替控件。



图6.63商与余数

2的幂次运算：如 n 端子连接的数是正值，那函数会将 x 端子的输入值乘以2的 n 次方（ 2^n ）。如果 n 端子为负值，那函数会将 x 端子的输入值除以 2^n 。比如将 n 设为+4，相当于 x 乘以16，如设为-4，相当于 x 除以16。这个函数比商与余数函数的效率更高。然而，尽量将 n 端子连接一个最小的数值。

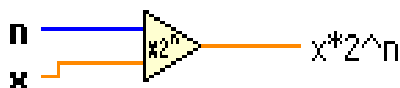


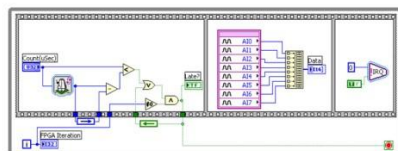
图6.64 2的幂运算函数

注:就发送数组数据来说，使用DMA比创建一个前面板数组指示器并使用FPGA读出/写入具有更高的优越性。可以使用数组将一系列同步采样数据发送到DMA缓冲区，然后再传送到主机上。只要不为数组创建前面板指示器，就可以用它将一系列数据点索引到DMA Write函数，还可以在给DMA缓冲区写数据的for循环上使用自动索引功能，因为编译器善于优化传入for循环的数组。

贴士：避免使用前面板数组进行数据传送

当优化代码在FPGA上的使用空间时，首先应该考虑优化前面板上的控件和指示器。每个前面板对象及其承载的数据都占用了一部分可观的FPGA空间。减少这些对象的数量以及前面板数组的大小，就可以显著得减小VI所需的FPGA空间。

FPGA



Host

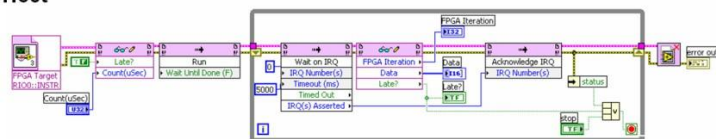
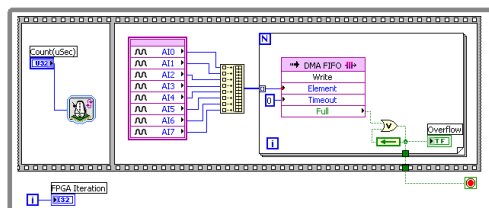


图6.65创建大型数组来储存数据并将其发送到主应用程序上

使用DMA向主机处理器传送模拟输入数组数据如图6.66所示，而不要创建大型数组来存储数据并将其传送至主机应用程序（见图6.65）。

FPGA



Host

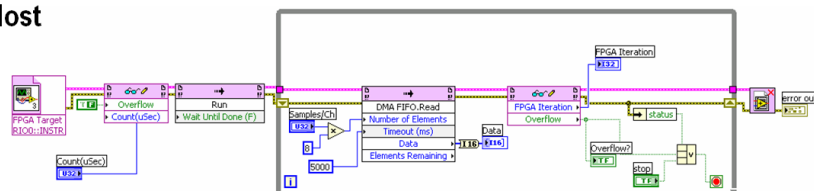


图6.66使用DMA向主机处理器传送模拟输入数组数据

贴士：使用DMA传送数据

DMA将数据储存在FPGA内存上，然后将这些数据高速传送到主机处理器的内存里，这个过程几乎不需要处理器的参与。使用DMA传送大型数据块，与使用前面板指示器及FPGA读出/写入相比，DMA所需的处理循环非常少。

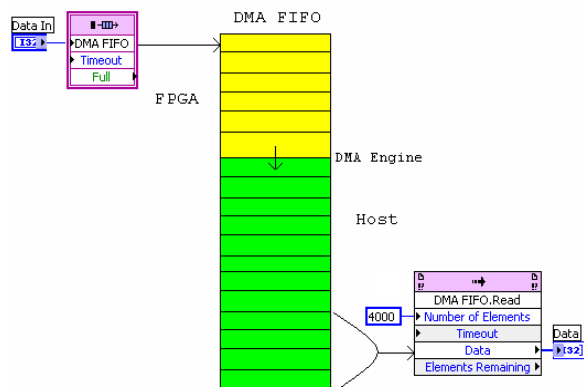


图6.67 DMA将数据储存在FPGA内存上，然后将这些数据高速传送到主机处理器的内存

使用DMA时的编程建议：

- 设置FPGA缓冲区的大小时，可以用默认的大小（1023）。创建大型的FPGA内存缓冲区通常不会带来好处。
- 通常应设置主机缓冲区大于默认值。默认情况下，主机缓冲区比FPGA缓冲区大两倍。实际上至少应将主机缓冲区设置为计划使用的单元数的两倍。
- 如果传送数组数据，Number of Elements（单元数）的输入应该始终为数组大小的整数倍。比如，如果传送一个8个元素的数组，Number of Elements（单元数）就应该为8的整数倍（比如80，每个单元分配10个采样点）。
- 每个DMA都已经被占用，所以通常读取大型的数据块是更好的做法。DMA FIFO.Read函数将自动等待，直到需求的Number of Elements变为可用。

- 在CompactRIO上将16位的通道数据打包为一个U32(因为DMA用U32数据类型)通常不会带来好处，因为PCI 总线用很高的带宽来传送DMA数据，这样就很可能远远用完所有的总线带宽。相反，在传送数流方面，通常处理器是技术瓶颈。在FPGA中打包数据就意味着还需要在处理器上解压数据，这就增加了处理器的内存开销。一般来说，即使采集16位数据，也需要以U32来传送每个通道的数据。
- DMA FIFO Write 函数上的Full输出实际上是个错误指示器。在正常操作下，不可能会发生错误。所以如果出现错误，NI 推荐停止应用程序并再重启之前重新配置FPGA。

贴士：使用最小数据类型

记住，在LabVIEW FPGA中编程时，一定要使用最小数据类型。比如使用一个32位整数（I32）来索引选择结构就会造成资源浪费，因为编写的程序代码不会拥有20亿不同的选择分支。通常使用一个无符号的8位整型（U8）就能满足需求，因为它能提供多达256个不同的选择分支。

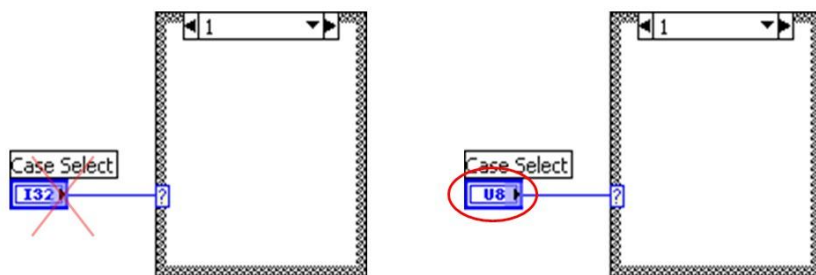


图6.68使用一个无符号的8位整型（U8）就来索引选择结构，因为它能提供高达256个不同的选择分支

贴士：优化大小

这个FPGA应用程序因过大而不能编译，为什么？

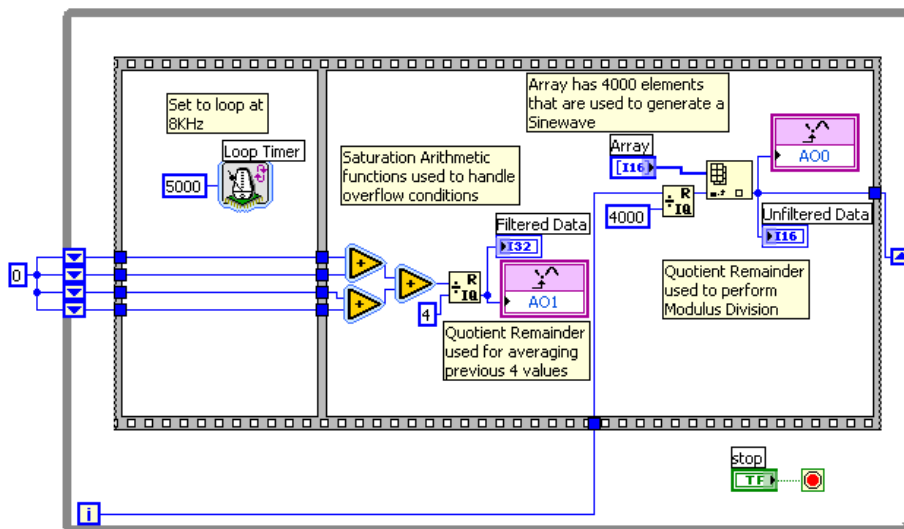


图6.69 这个VI因过大而不能编译

这个应用程序使用一个数组来存储正弦波数据。使用索引来得到数组元素。另外前4个点被存储在移位寄存器内，并取其平均值。这个VI因过大而无法编译。那么如何优化程序代码呢？

程序中出现了四个Gate hogs（逻辑门浪费者）：大型前面板数组，商与余数函数。

为了改进应用程序，将数组替换为LUT，如图6.70。

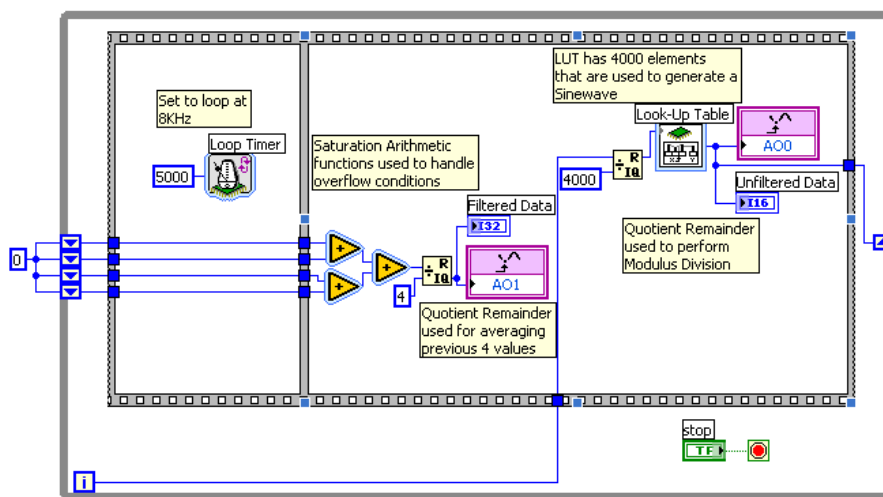


图6.70 将数组替换为LUT来优化程序

完成个变换后，就可以编译这个程序了。它仅占用1M gate FPGA的18%。能更进一步优化程序吗？

下一步移除两个商与余数函数。其中一个用于LUT索引的函数，用移位寄存器来代替，这是一个在FPGA常见的技术。另一个函数则用2的幂次运算（Scale By Power of 2）来代替。因为2的幂运算有一个输入端子为常量，它占用的FPGA空间非常小。注：比例 2^{-2} 相当于除以4。

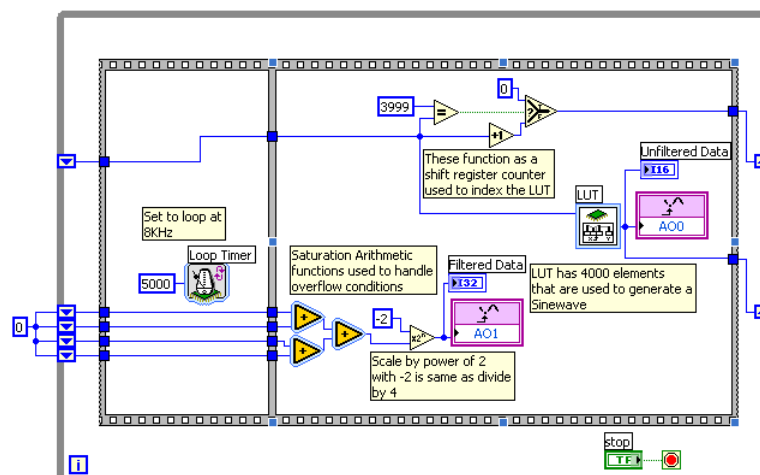


图6.71 移除两个商与余数函数来进一步优化程序

现在应用程序仅占9%的FPGA逻辑门。

贴士：优化FPGA应用程序的其他技巧

要想了解关于本话题更多的信息，可以参考NI公司的“[Optimizing FPGA VI for Speed and Size](#)”白皮书。这本书里所包含的用于优化FPGA应用程序的技巧超过了10个。

优化技巧	FPGA速度	FPGA大小
减少组合路径	✓	
适当的时候使用流水线技术	✓	
使用单周定时循环	✓	✓
使用并行操作	✓	
选择适当的仲裁选性	✓	✓
使用不可重用的子VI		✓

使用可重用的子VI	✓	
限制前面板上的对象数量，比如数组		✓
尽量使用最小的数据类型	✓	✓
尽量避免使用大型VI和函数	✓	✓
使用握手信号来安排定时	✓	✓

表6.3优化FPGA应用程序的技巧

第7章

为CompactRIO创建网络用户界面

使用LabVIEW建立用户界面及HMI

本文将介绍一个软件设计架构，建立的操作界面具有可在不同HMI页之间循环的可扩展导航引擎。

你可以使用此架构为任何HMI硬件目标建立基于LabVIEW的HMI，包括运行Windows XP Embedded OS 的NI TPC-2512触控面板计算机，或运行Windows CE OS的NI TPC-2106及运行Windows XP OS的NI PPC-2115面板PC。LabVIEW是完整的编程语言，为各种开发任务提供解决方案，范围包括HMI/SCADA系统至可靠性及确定性控制应用。LabVIEW Touch Panel Module提供图形化编程界面，可用于在Windows开发环境下开发HMI并将其配置到National Instruments触控面板计算机（TPC）或任意运行Windows CE的HMI。本文为工程师开发基于Windows VistaXP及Windows CE的HMI提供框架。

基本HMI架构背景

HMI可与你需要的功能一样简单或复杂。选择正确的软件架构决定了HMI的功能及其扩展并适应于未来技术的能力。基本的HMI包括三个主要例程：

- 1. 初始化及停止（内务）例程
- 2. I/O扫描（记忆表、I/O及通讯驱动）循环
- 3. 导航（用户界面）循环

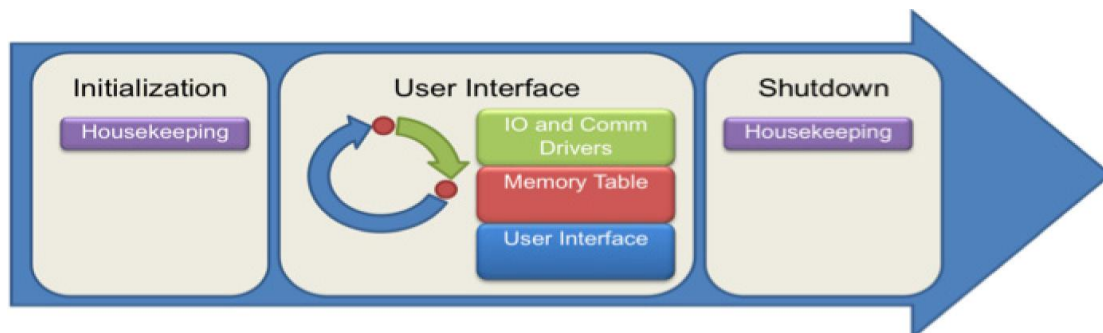


图7.1. 典型HMI的例程

初始化及停止

执行I/O扫描循环及导航循环前，HMI需要执行初始化例程。这个初始化例程设置全部输入控件、显示控件、内部变量及硬件（控制器）与默认状态间的通讯变量。你可以添加更多逻辑，为诸如文件记录的操作准备HMI。停止系统前，停止任务关闭任何参考并执行附加任务，例如记录错误文件。

I/O扫描循环

整个HMI架构与CompactRIO架构十分相似，均使用记忆表和基于命令的架构在任务间传递数据。

记忆表

本架构中，I/O扫描循环从网络读入并写出数据并将其放入记忆表。其它例如UI页的任务由记忆表读入和写出数据。利用单进程共享变量创建记忆表，I/O扫描由网络发布的共享变量读入数据。这将使系统开销最小化，并创建扩展性更强的应用。

基于命令的架构

I/O扫描同时运行一个命令处理程序，为real-time系统将该命令转换为网络命令。为保证操作的可靠性，HMI上的每个用户命令须要可靠地发送至real-time控制器。为达到此目的，使用一个具有命令FIFO的基于命令的架构。



图7.2. 采用网络变量的简单命令架构

启用缓存的网络发布的共享变量创建一个FIFO，并作为命令从HMI发至控制器的传输层。使用LabVIEW队列将命令从用户界面页发送至I/O Scan中的命令处理程序。

有关网络命令，以及接收来自HMI的分析命令而建立的工作架构的更多信息，请参考“通讯”部分。

导航循环

导航循环负责管理UI（显示页）。它包括UI页及导航引擎，帮助你在HMI中可用的不同页之间循环。

UI页

每个UI页均为一个LabVIEW VI。UI页包括以下几个最常用的组件：导航按钮，动作按钮，数值型输出控件，图表，布尔型输入控件及输出控件，图像。UI页读取单进程共享变量上显示的值。

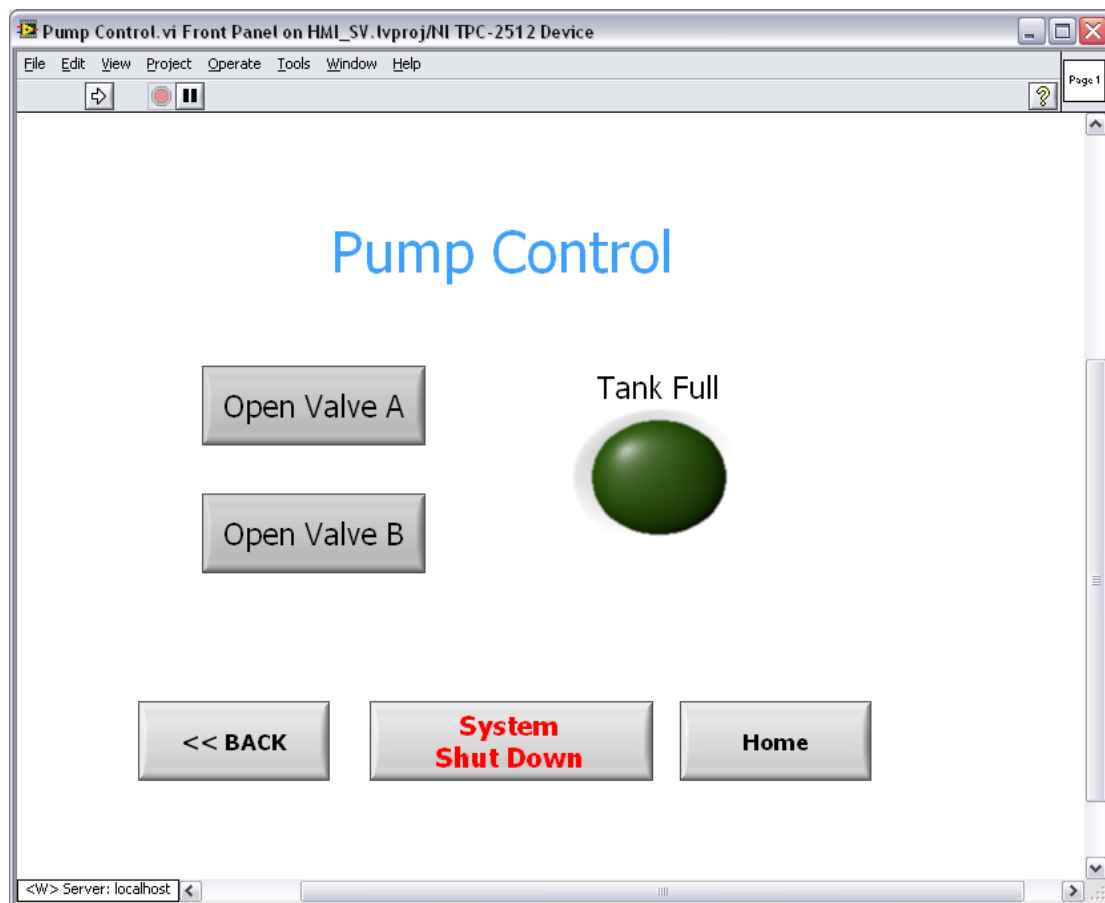


图7.3. LabVIEW中典型UI页示例

用户动作既是导航引擎的命令，又是实时控制器的命令，例如按动按钮。你也可以使用LabVIEW事件结构获得命令并将其发送至导航引擎，或使用队列传递至命令处理器，随即放入网络。为有助于创建事件驱动UI，你可以使用标准LabVIEW UI-handling模板。通过传递数据至队列，将UI事件转化为适当的命令，进而被命令处理器处理。

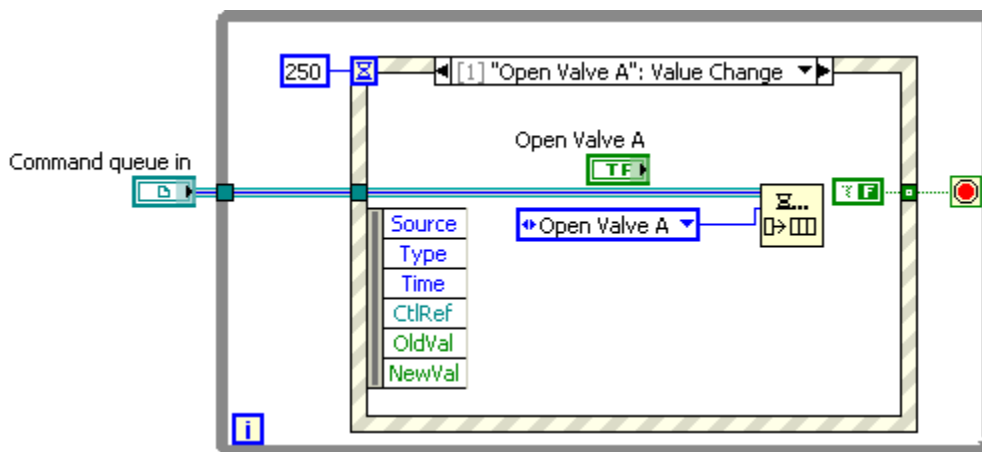


图7.4. 将数据传递至命令队列的用户界面页

导航引擎

导航引擎是一种子架构，它是导航循环的一部分并负责处理UI页。它是使用while循环和事件结构建立的LabVIEW状态机。每个事件装入一个UI页，它是一个LabVIEW VI，当被调用时设置为开。

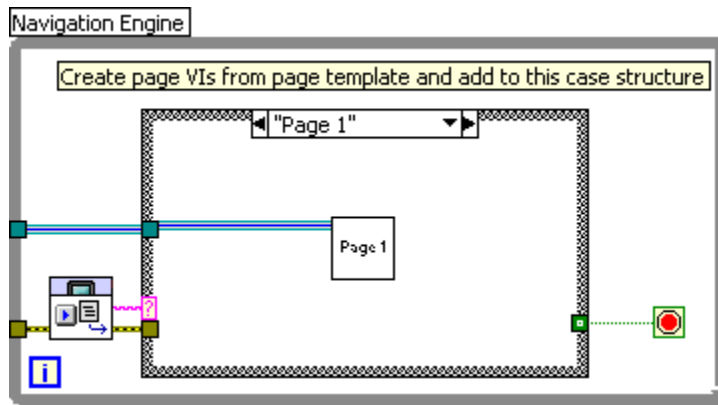


图7.5. 导航循环的框图

导航引擎使用Touch Panel Navigation Palette 中的VI。

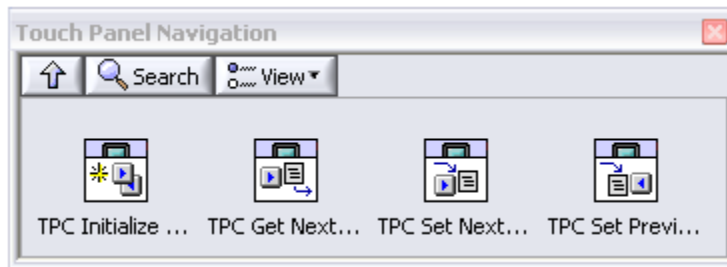


图7.6. 导航面板

TPC Initialize Navigation

此VI通过设置导航历史深度和所显示的首页的名字，将导航引擎初始化。

TPC Get Next Page

此VI返回下一页的名称，并将其发送至HMI导航引擎中的事件结构。

TPC Set Next Page

此VI设置下一页的显示名称。在HMI页内使用它来支持导航按钮。

TPC Set Previous Navigation Page

此VI返回来自页历史的前一页的名称。在HMI页内使用它来支持“back”导航按钮的操作。

页的状态和历史存储在一个这些VI均可进入的功能全局变量中。

Windows XP, XP Embedded及CE Operating 系统的基本HMI架构

为获得全功能HMI应用，推荐使用Windows XP和XP Embedded。XP和XP Embedded上的图形处理能力和LabVIEW的灵活性更强大。此外，使用Windows XP更易于结合其它I/O或与使用其它机制的设备通讯，例如OPC。但是对于提供简单扩展而言，全部架构对于Windows XP, XP Embedded及CE 平台是相同的。

为阐释这一架构，建立一个基本HMI应用。这个应用包括两个UI页-一个起始页和一个泵控制页。每个UI页都有连至其它页的导航按钮。除导航按钮之外，UI页都有控制和指示器，通过I/O扫描循环上网络发布的共享变量读入来自控制器的值，并局部存储在单进程共享变量中。你可以使用这一框架添加更多页以适应你的应用。

I/O表格

所有过程数据都存储在HMI上的记忆表中。这个记忆表利用单进程共享变量建立，可由IO扫描或独立的UI页读入和写出。为了创建I/O表格，完成以下操作：

- 添加触控面板目标至LabVIEW Project。
- 在触控面板目标下，建立一个subVI文件夹。在此文件夹内，创建一个I/O Memory Table 文件夹。
- 右击文件夹并选择New»Variable。
- 以单进程变量创建变量。因为触控面板不运行实时操作系统，这些变量并没有启用的实时FIFO。

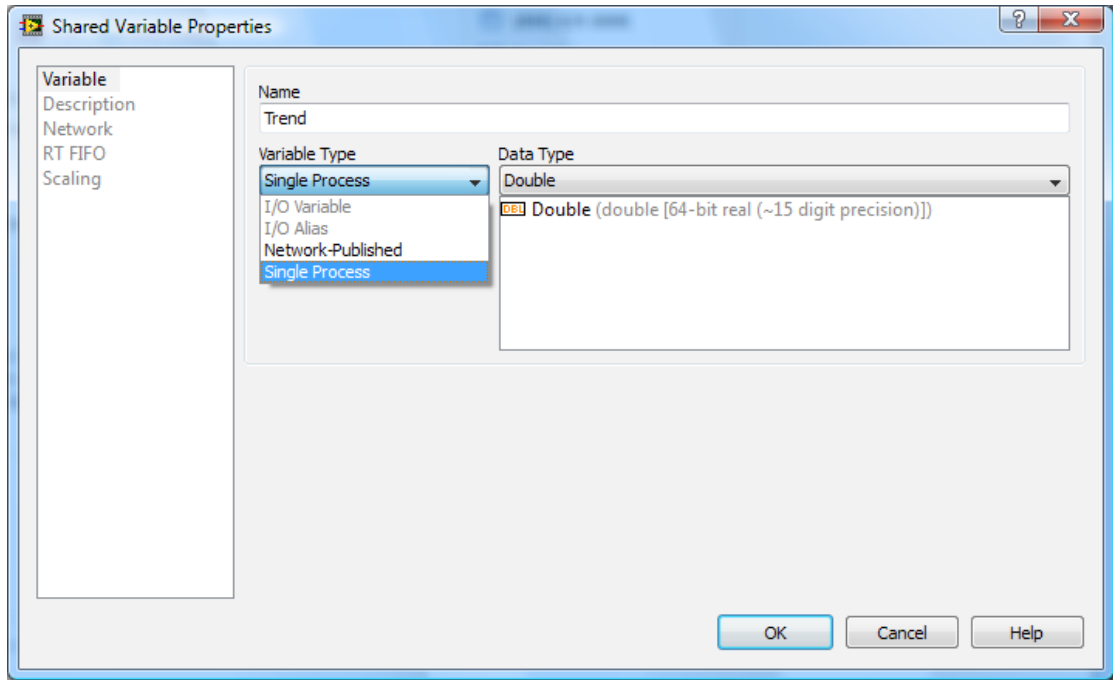


图7.7. 创建单进程变量

- 保存你的project和library。

初始化任务

当HMI启动时，这一任务将控制，指示器及内部变量设为默认状态。它也为内部命令创建队列。

1. 打开一个顶层VI并创建平铺式顺序结构。
2. 在第一个框内创建一个subVI并保存为Default HMI。
3. 打开subVI并通过写入单进程共享变量为记忆表给定默认值。保存变化。

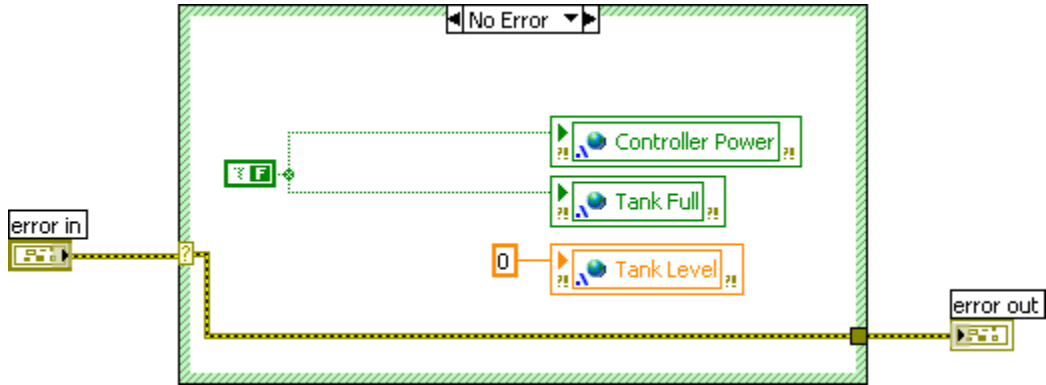


图7.8. 设定单进程共享变量的默认值

4. 命令队列也在初始化任务中创建，因此任何需要发送至控制器的命令均排队等候，而不错过任何命令。为使编程扩展性更好，使数据类型以type def enum队列??。这样，如果你需要添加命令，变化将通过你的全部代码自动传播。

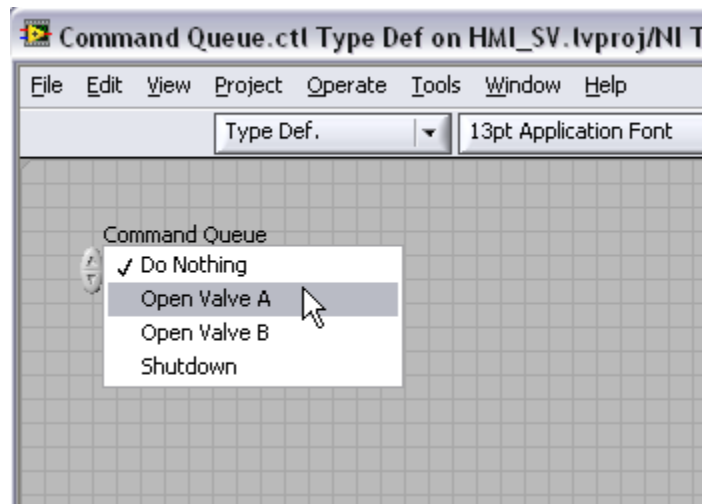


图7.9. type def enum提供可扩展性

5. 在第一个框内再创建一个命令队列，并连线至刚才创建的type def enum

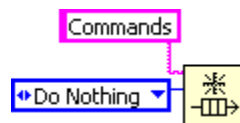


图7.10. 为命令创建一个Enum队列

I/O扫描循环

I/O扫描循环在网络和记忆表以及命令队列之间传递数据和命令。为创建扫描循环，完成以下动作：

1. 在顺序结构的第二框中，在图上拖放一个while循环。使用Wait(ms)函数来配置匹配你的应用需求的通讯循环率。
2. 在图中创建两个subVI：Network Communication.vi 和 HMI Command Handler.vi。

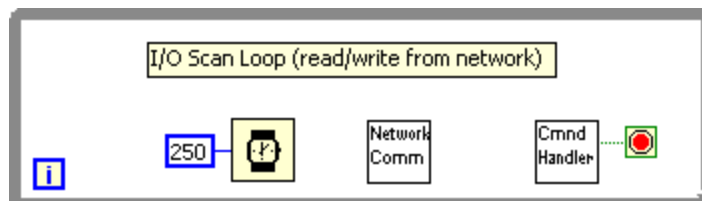


图7.11. I/O扫描循环

3. 打开Network Communication subVI。在框图内，放置单进程共享变量，并将其写入适当的网络发布的共享变量中（寄存在CompactRIO控制器上）。本例中，执行简单错误检查，如果网络变量出现任何错误或警告，你不要在记忆表中放入任何数据（单进程共享变量）。

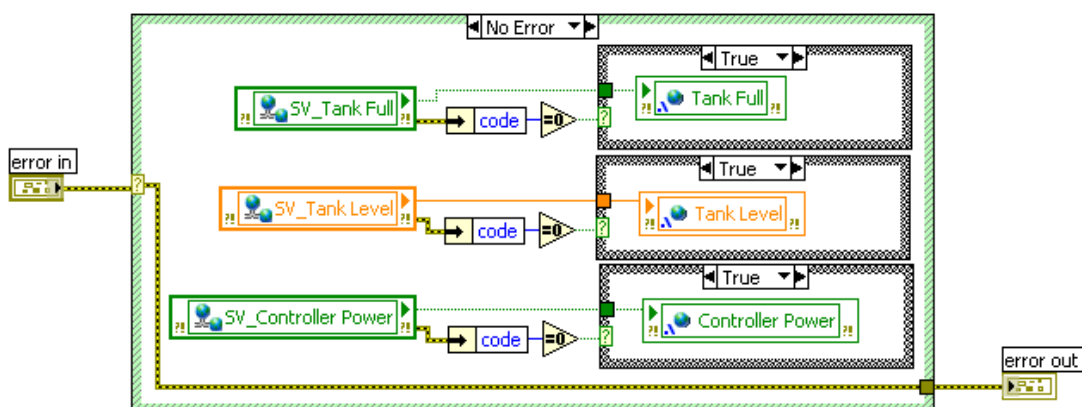


图7.12. 网络数据被写入记忆表

4. 打开HMI Command Handler subVI。此VI负责为实时控制器发送任意指令至网络。UI页中队列的命令出列，使用“Command”网络变量分别发送至控制器的命令寄存在实时控制器上。创建图7.13所示的框图，包括一个发送每条指令的case及一个“Do Nothing”命令的默认case。

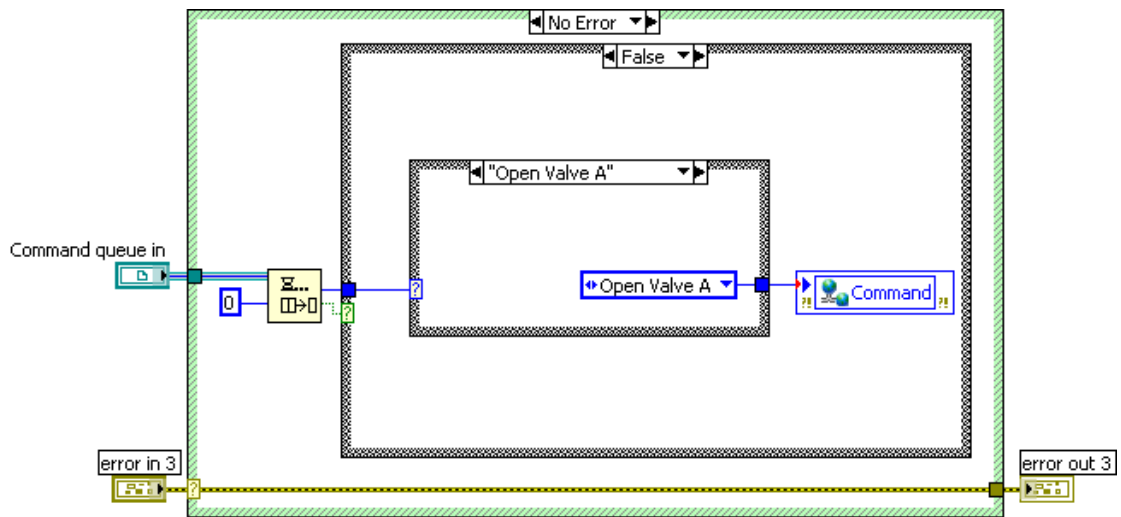


图7.13. Command Handler SubVI

5. 保存并关闭两个subVI。
6. 连线初始化subVI及扫描循环。本应用中，命令处理器在接到关机命令时也很停止循环。

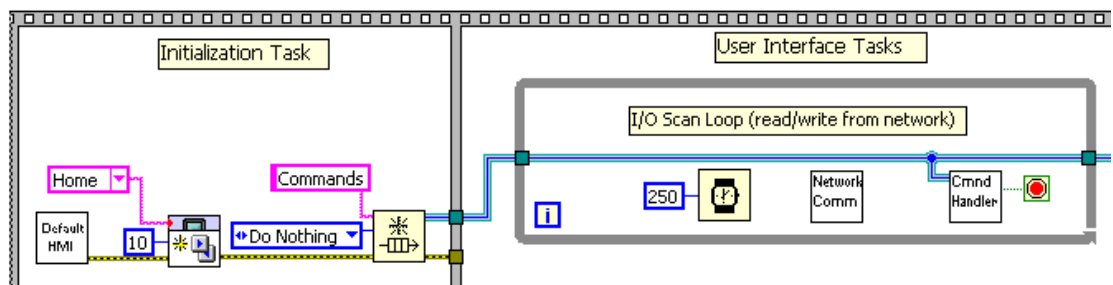


图7.14. I/O扫描及初始化任务

导航循环

导航循环由导航引擎和UI页组成。

导航引擎

使用导航引擎状态机架来管理UI页。这个状态机使用一个API，它由LabVIEW Touch Panel Module 8.6或之后版本发送。如果你正使用以前的版本，你可以在ni.com下载此VI。

编写导航引擎最简单的方法是从实例中拷贝代码。

1. 打开位于C:\Program Files\National Instruments\LabVIEW 8.6\examples\TouchPanel\navigation\Design Pattern Template的模板VI。

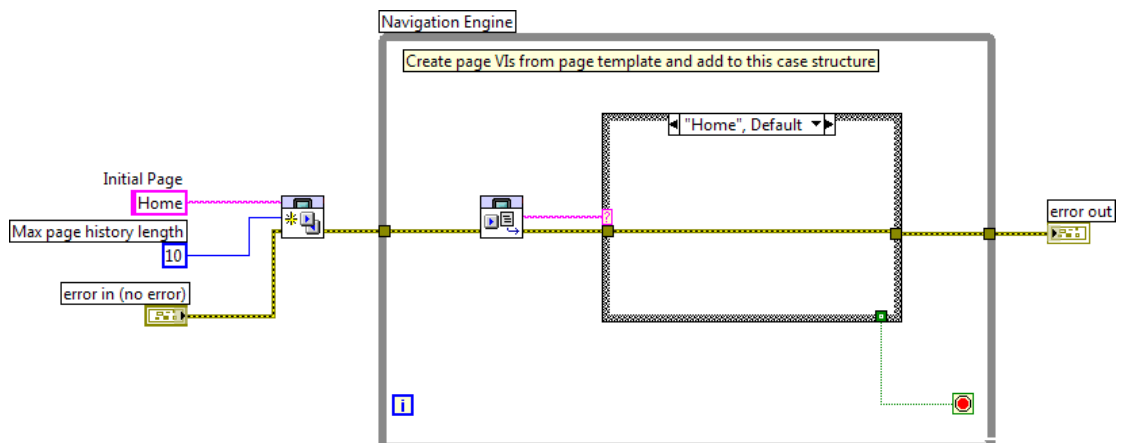


图7.15. 模板VI

- 由模板拷贝整个框图并粘贴在第二个sequence内，作为扫描循环的并行循环。
- 为每个UI页创建一个case，并增加一个用于关机case。

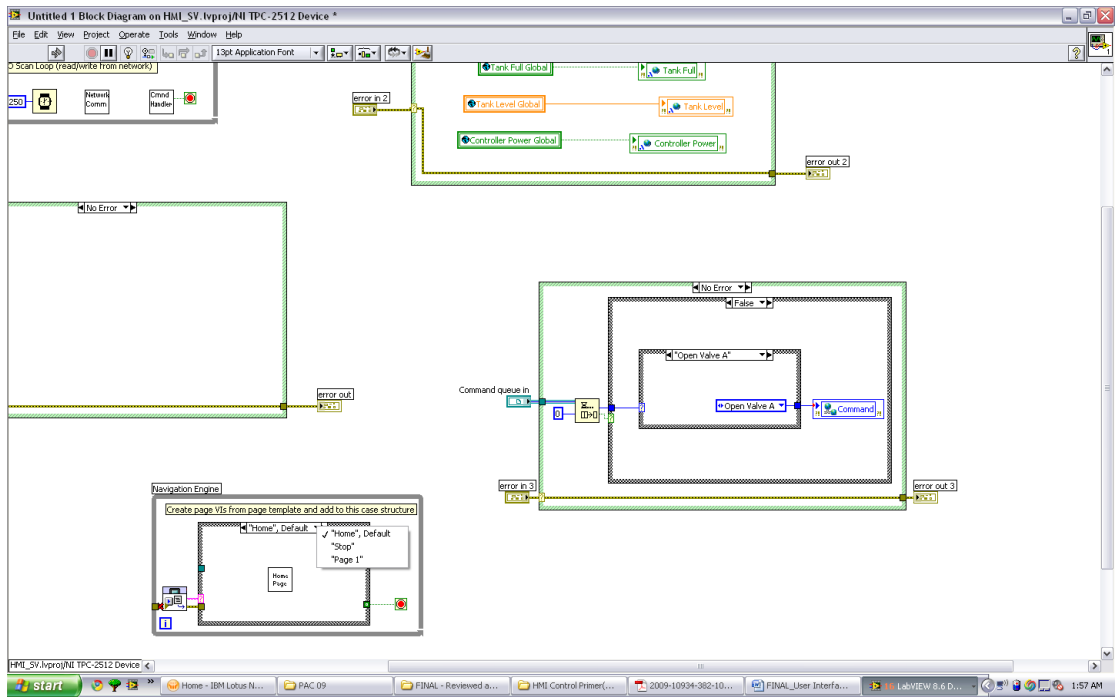


图7.16. 添加cases至你的HMI

- 确定当HMI首次开始时你想打开的页。本应用中，“Home”中的UI页是充当起始页。使用TPC Initialize Navigation.vi设置起始页作为进入while循环前的起始页。

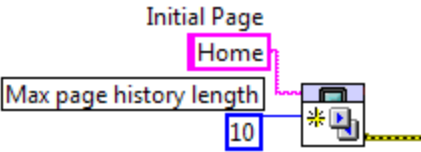


图7.17. TPC Initialize Navigation.vi

- 进入停止case，将“Shutdown”加入至在初始化循环中创建的命令队列。这将关闭控制器和HMI。

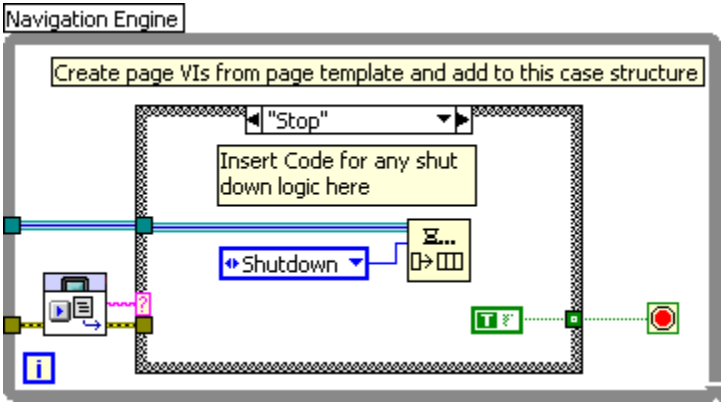


图7.18. 在命令队列中加入关机命令

此时完成了导航引擎的开发。下一步是开发UI页。

UI页

建立UI页时首先要做的是设置前面板尺寸。因为前面板可能大于HMI显示，所以将你的UI元素限制到指定边界是很重要的。例如，当使用NI TPC-2512 HMI时，LabVIEW前面板尺寸需要与800x600 像素的TPC分辨率匹配。

- 打开一个新VI并根据你的硬件目标设置尺寸。本例中，VI为TPC-2512目标而设。

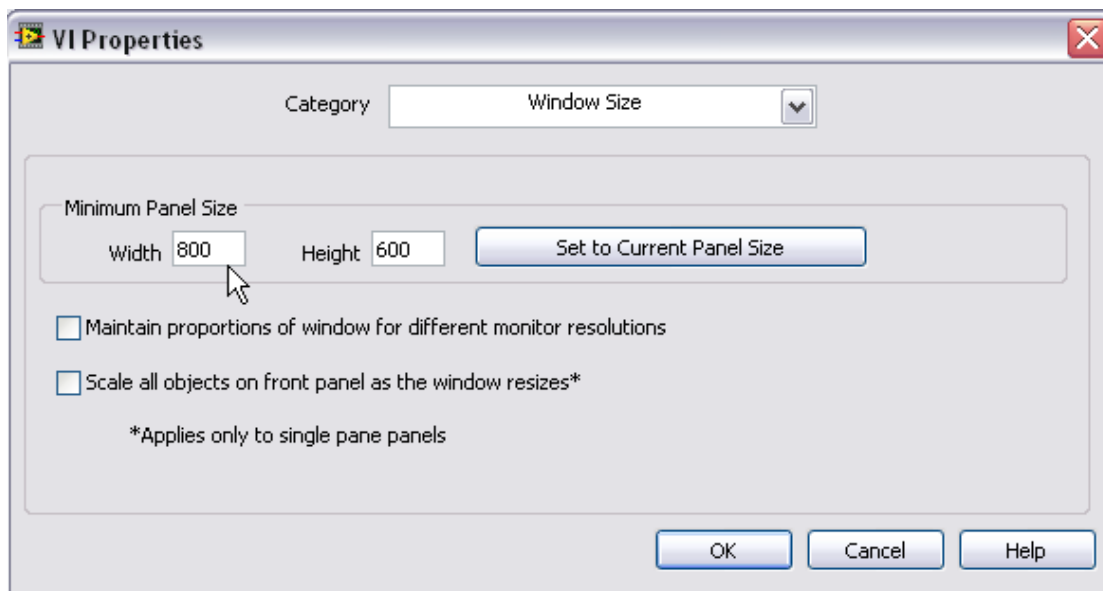


图7.19. 根据触控面板分辨率设置前面板尺寸

2. 下一步是为由UI页控制所执行的函数选择UI元素。本例中泵控制页有一些布尔控制、指示及导航按钮。

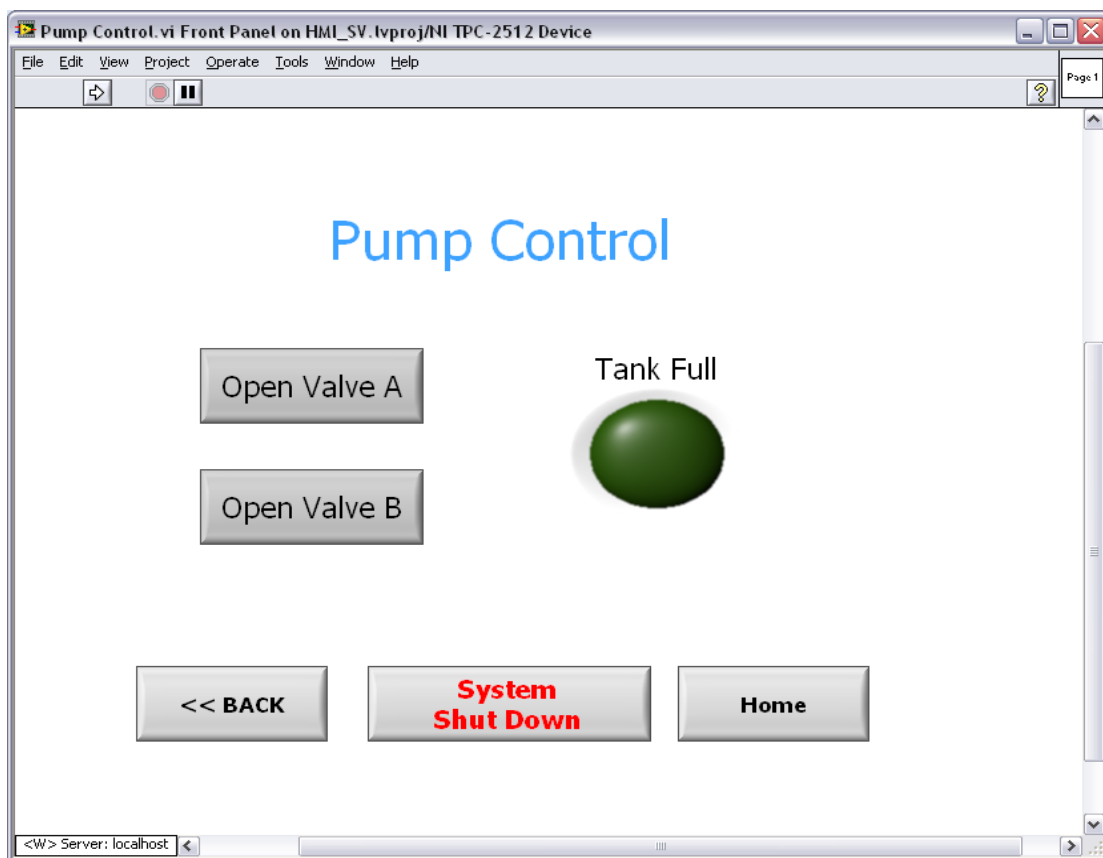


图7.20. UI页实例

3. Back导航按钮使你回到最后访问页，Home按钮则回到前一部分组合的导航引擎的“home” case中的UI页。
4. 在UI页框图中，放置一个while循环和一个事件结构。将250连至事件结构左上角的timeout输入。这意味着如果没有检测到其它用户事件，每隔250 ms执行一次timeout case。

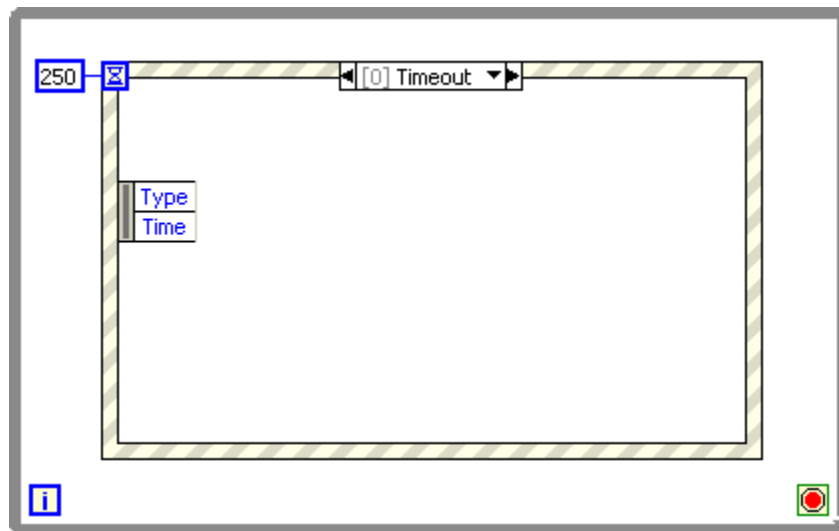


图7.21. 如果没有检测到其它用户事件，每隔250 ms执行一次timeout case

5. timeout事件case由单进程共享变量读入/写出数据至该页的控制和指示。放置“Tank Full”单进程共享变量并连线至各自的控制。注意，“False”常数要连至while循环的停止终端。这将确保case执行时循环的连续运行。

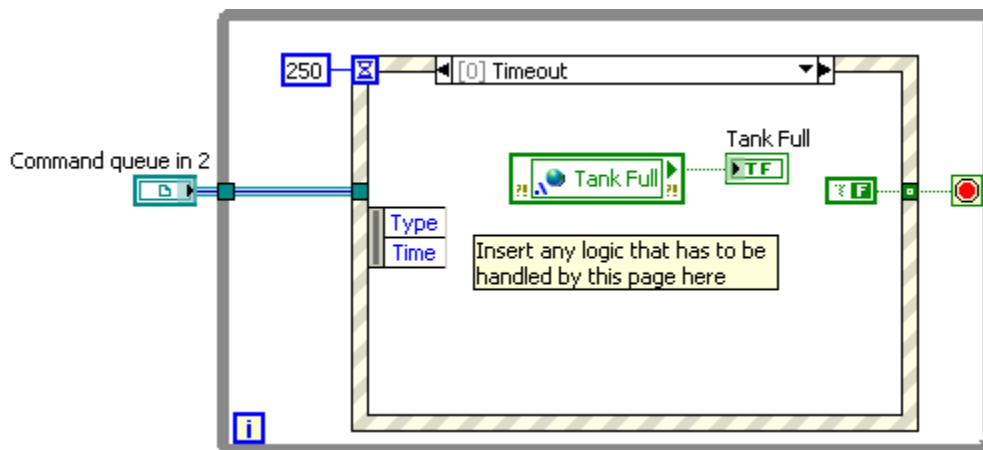


图7.22. “False”常数要连至while循环的停止终端以确保case执行时循环的连续运行

6. 现在，“Open Valve A”和“Open Valve B”命令需要加入在初始化循环中创建的队列。为使党按钮值发生变化时发送命令，添加事件case。在相应的事件case中队列指定的命令。确保在相同的事件case中读入按钮控制以使按钮重置（如果按钮为封锁机械动作配置）。

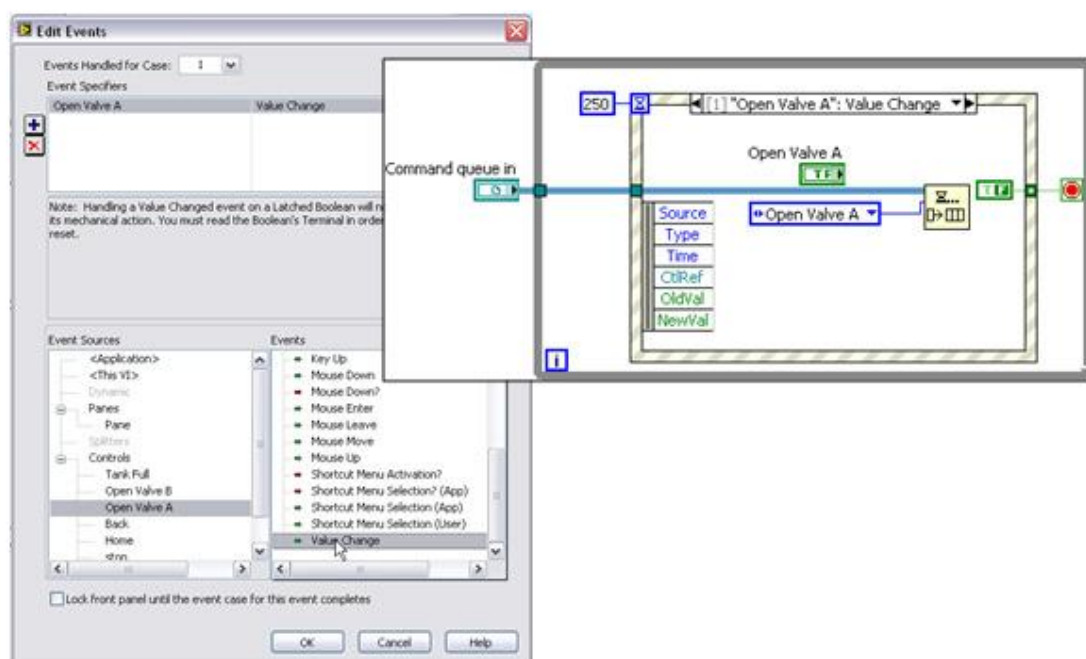


图7.23. “Open Valve A”命令加入命令队列

7. 为每个命令或按钮添加事件case。
8. 在Home event case中，放置Home Boolean按钮及Touch Panel Navigation Palette中的TPC Set Navigation Page，并连线至名为Home的字符串常量。这将使被访问的下一页设置为导航引擎“Home” case中的VI。连线“True”常量至此事件case中的while循环的停止终端。这将停止循环，关闭此UI，并返回导航引擎，以使其可以打开用户设置的下一页。

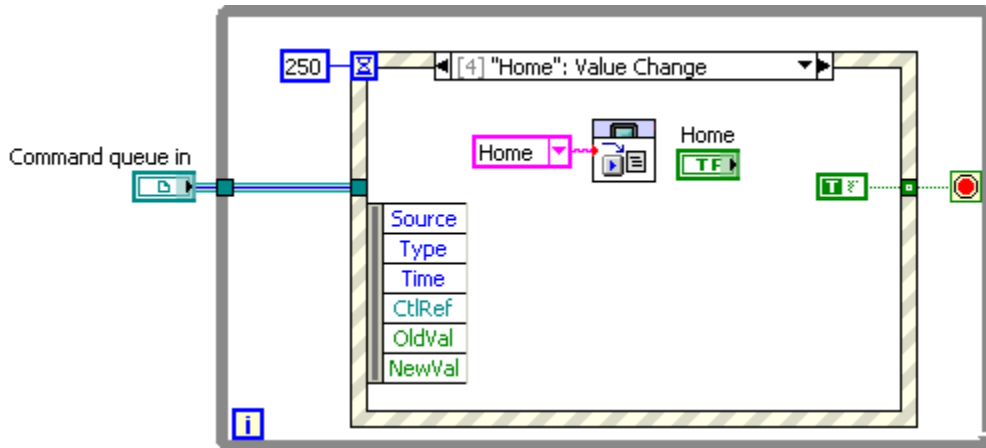


图7.24. 连线至停止终端的“True”常量终止循环并返回导航引擎，以使其打开下一页

9. 对于Back按钮的事件case，使用TPC Set Previous Navigation Page.vi来退回至上一访问页并停止VI。

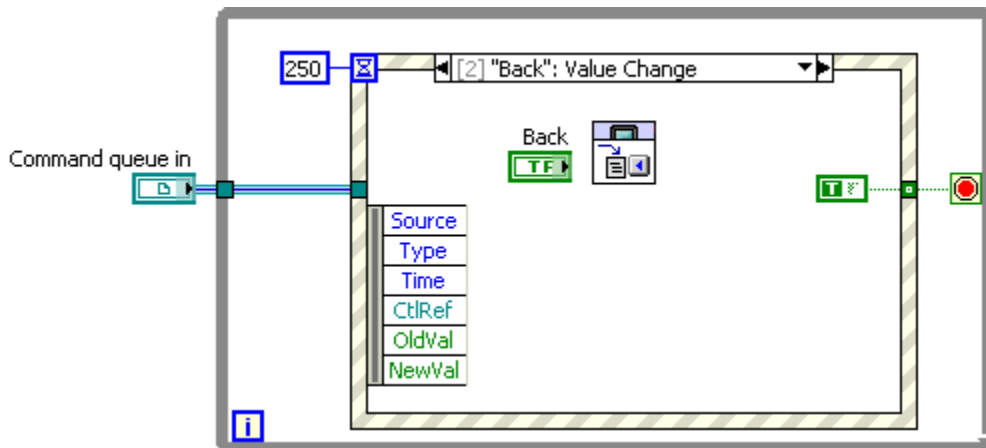


图7.25. 使用TPC Set Previous Navigation Page.vi返回至前一访问页

10. 最后，对于系统关闭case，将“stop” case连线至TPC Set Next Navigation Page.vi。

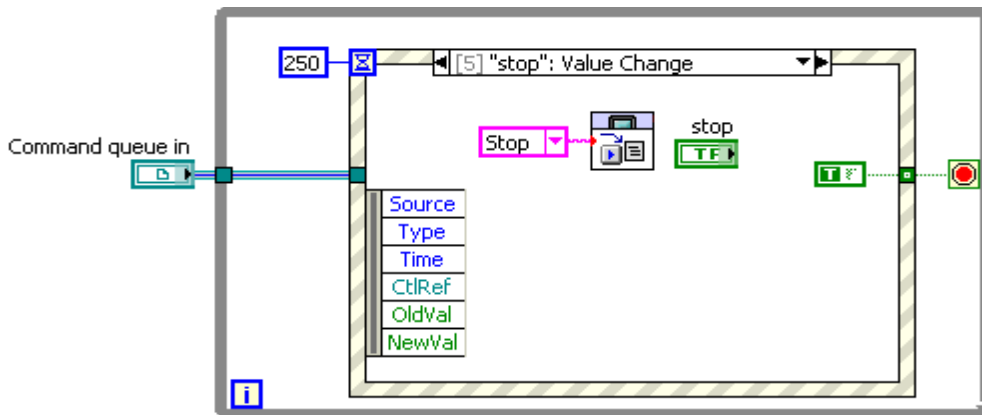


图7.26. 连线“Stop” Case至TPC Set Next Navigation页

11. 保存此VI并放置在“Page 1”的导航引擎选择结构中。这将确保当用户按下按钮打开泵控制页时打开该页。

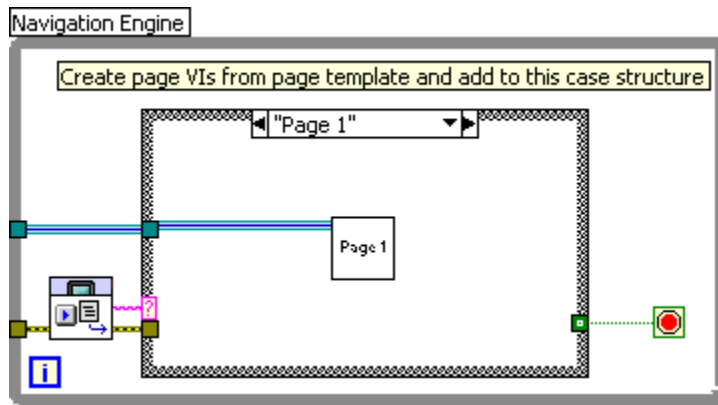


图7.27. 确保当用户按下按钮时泵控制页打开

12. 右击subVI，进入SubVI Node Setup并检查选项，则前面板当调用时出现，关闭时消失。

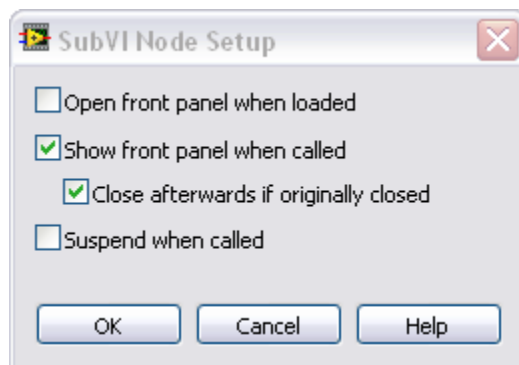


图7.28. 检查SubVI选项

13. 重复本部分的步骤来为其它UI页构建VI。
14. 这是最终的应用。添加任意需要的关机逻辑，例如关闭队列。

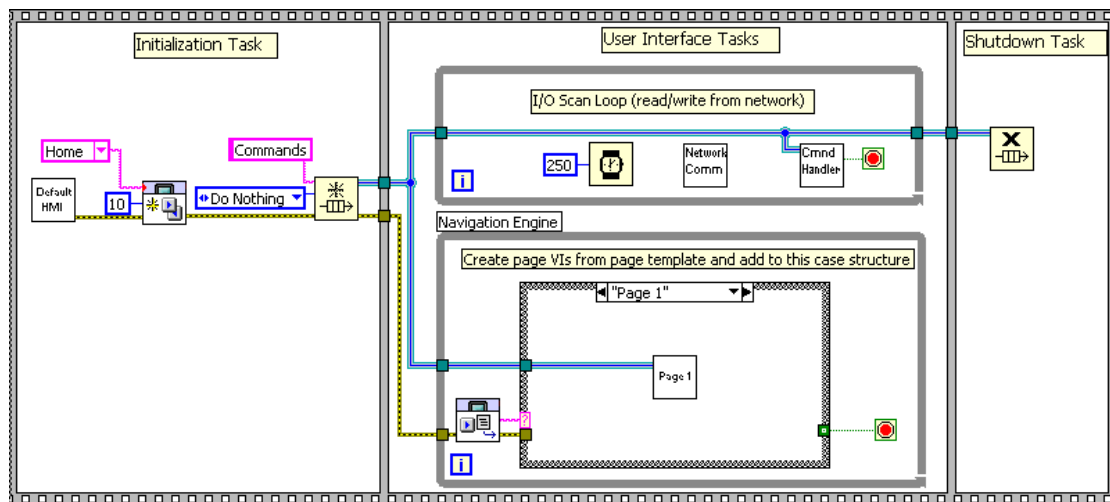


图7.29. 顺序结构中的扫描循环和导航引擎初始化

入门指南-修改实例

开始此项设计的最简便方法是修改一个现有的实例。修改之前的实例遵循以下主要步骤：

步骤1.修改记忆表

1. 编辑IO_Table.lvlib，它包含单进程共享变量，显示你需要从UI页读入或写出的进程数据。

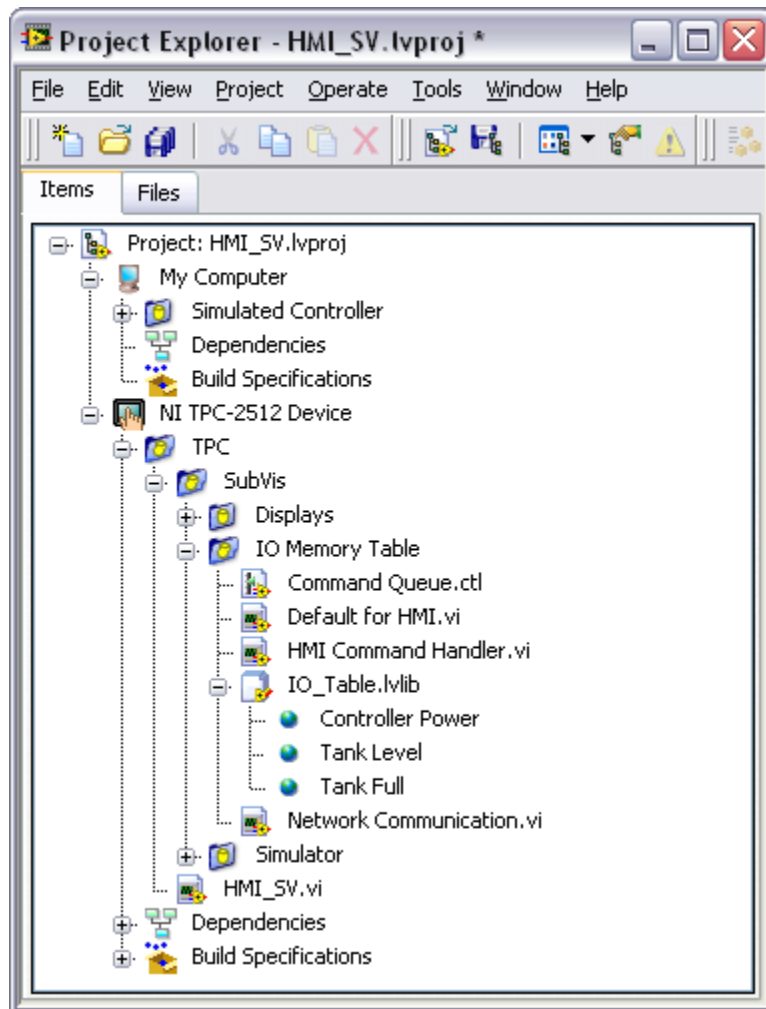


图7.30. 添加单进程共享变量至IO_Table.lvlib

2. 修改Default HMI.vi来为控制和指示写入的默认值。

步骤2. 修改命令类型定义

1. 编辑命令枚举型定义来代表应用中需要传递的命令

步骤3. 编辑I/O扫描循环

1. 修改I/O扫描循环，从适当的网络变量读取和写入并将数据传递至记忆表（单进程共享变量）。
2. 修改命令控制器以读取你的UI命令，并发送适当的网络命令至寄存在CompactRIO上的命令变量。

步骤4. 编辑导航循环

1. 为导航引擎添加条件结构以包含更多页面
2. 为每个UI页面创建新的VI，并将它们添加到合适的条件页面中

第八章

系统的配置与复制

系统的配置

使用LabVIEW对实时目标和触摸板目标的开发是在Windows PC上进行的。为了运行嵌入在这些目标中的程序代码，必须对系统进行配置。实时控制器和触摸板更像PC一样，拥有易失性存储器和非易失性存储器。在配置程序的过程中，可以选择易失性存储器或者非易失性存储器。

将程序配置在易失性存储器中

如果将应用程序配置在目标的易失性存储器中，重启电源后程序将不复存在。在开发应用程序和测试代码时这种配置是非常有用的。

将程序配置在非易失性存储器

如果将应用程序配置在目标的非易失性存储器上，重启电源后程序仍然保留在目标上。还可以设置保存在非易失性存储器上的应用程序，当目标启动时，应用程序将自动启动。如果已经完成应用程序的开发和测试，并想将这个应用程序做成单机嵌入式系统时，这种配置方式是非常有用的。

将系统配置在CompactRIO上

将LabVIEW程序配置在易失性存储器上

当将系统配置在CompactRIO控制器的非易失性存储器上时，LabVIEW将在以太网上搜寻所有必要的文件，并将它们下载到CompactRIO控制器上。通过下列必要的步骤配置应用系统：

- 将compactrio控制器放入到LabVIEW中
- 在控制器上打开一个VI
- 开启运行按钮

LabVIEW检查VI和子VI是否保存，将程序代码配置在CompactRIO控制器的非易失性存储器上，并运行嵌入的程序。

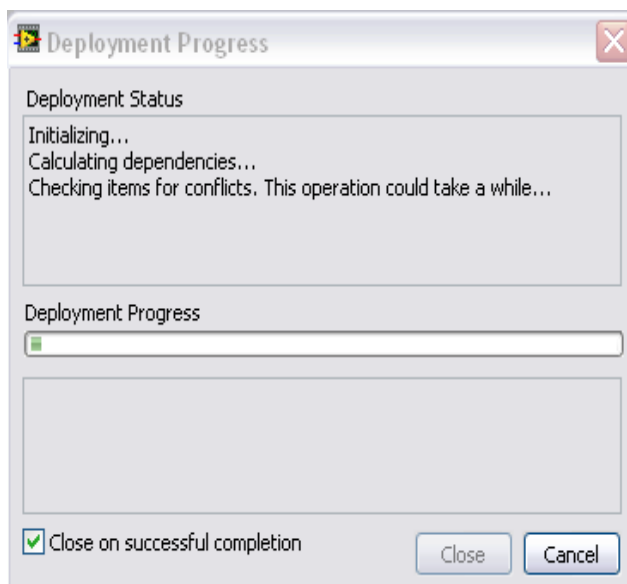


图8.1 LabVIEW将程序配置在控制器的非易失性存储器上

将LabVIEW程序配置在非易失性存储器上

一旦完成了程序的开发和调试，就可以将程序配置在控制器的非易失性存储器上。这样通过重复启动和配置系，系统就可以启动后运行。为了将应用程序配置在非易失性存储器上，首先需要将VI生成一个可执行文件。

将VI生成可执行文件

在LabVIEW工程中可以将一个VI生成一个可执行的实时应用程序。为了生成可执行的实时应用程序，在LabVIEW工程浏览器中的实时目标下创建一个生成规范。通过右击生成规范，就可以看到实时程序、源代码发布和压缩包等选项。

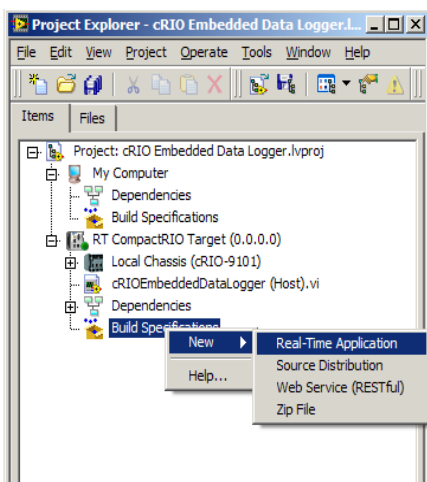


图8.2 创建实时应用程序的生成规范

选择**实时应用程序**后，会弹出一个对话框。对话框上的信息和源文件选项是生成实时应用程序时最常用的两个选项。目标、源文件设置、高级和附加排除选项在生成实时应用程序时很少被使用。

信息选项包括规范名称、可执行文件名以及实时目标和计算机目的目录。可以修改表规范名称和计算机目的目录来匹配系统命名和文件格式。通常并不需要改变目标文件名或者目标目的目录。

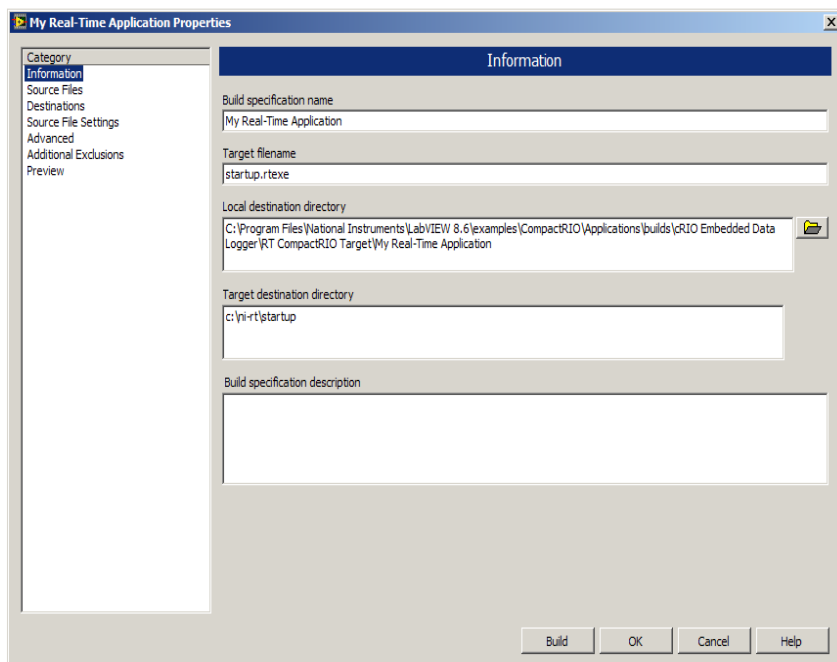


图8.3 实时应用程序属性上的信息选项

源文件选项被用来设置启动VI并包含附加VI或支持文件。从工程文件中选择顶层VI，将其设置成启动VI。对于大多数应用程序只需选择一个VI作为启动VI，不需要包含库文件或设置子VI为启动VI或将这些子VI加入到“始终包含”的文件列表中，除非这些VI是被动态

调用的。

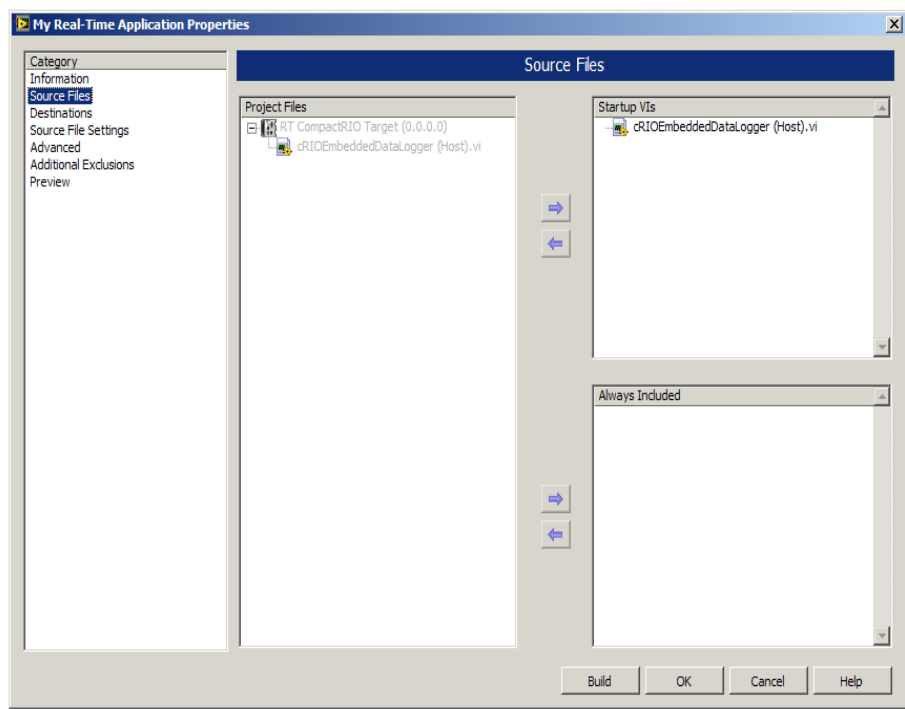


图8. 4 实时应用程序属性上的源文件选项（这个例子中cRIOEmbeddedDataLogger(主).vi作为启动 vi）

设置完必须类别列表中的所有选项后，单击OK键来保存生成规范或者直接点击生成键来生成规范。也可以右击保存生成规范选项并选择生成选项来生成应用程序。

生成应用程序后，一个可执行文件就被创建并被保存在开发设备硬件的本地目的目录里。

设置可执行的实时应用程序为启动运行

生成应用程序后，可以设置可执行文件在控制器开机后立即自动启动。右击实时应用程序选项（在生成规范下），选择设置成启动，这样就将可执行文件设置成了启动。将可执行文件配置在实时控制器后，控制器也就被设置成了在重启或者打开实时目标后自动启动应用程序。也可以将启动设置成非自动启动。

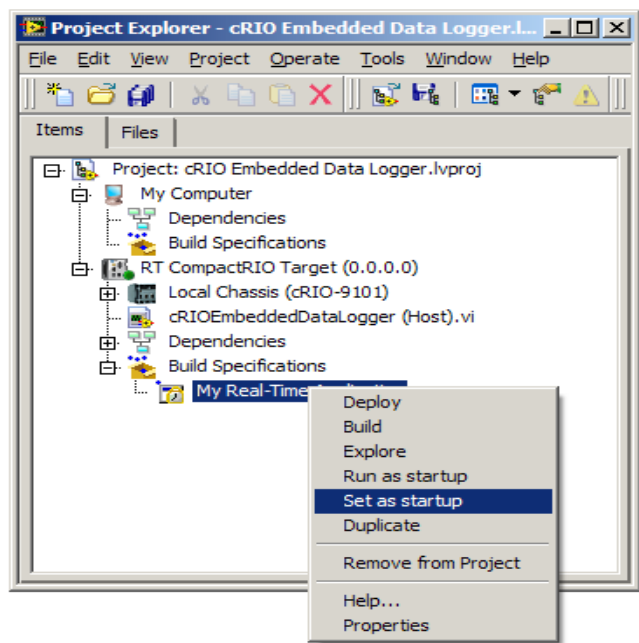


图8. 5 设置系统启动后实时应用程序自动启动

将可执行的实时应用程序配置在CompactRIO系统的非易失性存储器中

生成并配置完可执行文件后，需要将可执行文件和支持文件一起拷贝到CompactRIO的非易失性存储器中，并配置控制器使可执行文件启动后执行。右击实时应用程序选项，选择拷贝来拷贝文件和配置控制器。与此同时，LabVIEW将可执行文件拷贝在控制器的非易失性存储器中，并修改ni-rt.ini文件来设置可执行文件启动后执行。如果重新生成应用程序或者修改了应用程序的属性（比如将程序设置成启动后不运行），那么就需要重新拷贝实时应用程序，这些改变才会起作用。

有时候，想移动保存在实时目标上的应用程序。最简单的方法就是使用FTP进入实时目标删除可执行文件。如果使用默认设置，则可执行文件被放在NI-RT\Startup下的文件夹，文件夹名是信息选项页的目标文件名框提供的名字，扩展名是.rtexe。

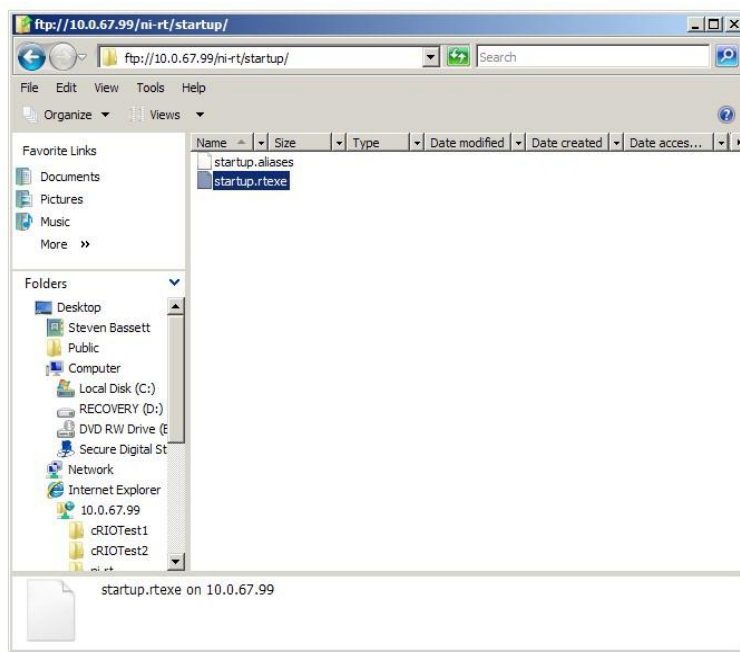


图8.6 从CompactRIO控制器中删除startup.rtexe

在触模板中配置应用程序

配置到触摸板的链接

虽然可以手工把生成的应用程序拷贝到触摸板设备中，但还是推荐使用以太网并允许LabVIEW工程自动下载应用程序。NI公司的触摸板都拥有一个叫做NI TPC Service的功能，这个功能允许LabVIEW工程在以太网上直接下载程序代码。右击LabVIEW工程中的触摸板目标，选择属性来配置链接。在总体类别页中，将链接设置成NI TPC Service，并输入触摸板的IP地址。测试链接来确定服务能够运行。

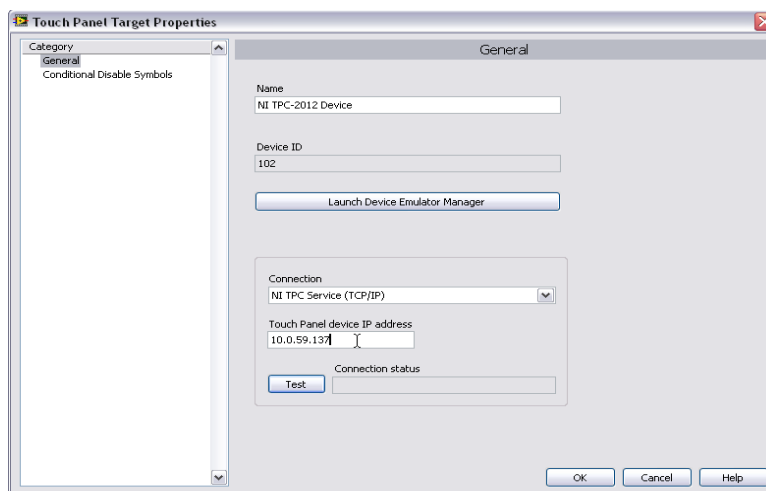


图8.7 通过以太网使用NI TPC Service与触摸板链接

选择开始菜单下的运行命令，在弹出的窗口上输入cmd，就进入命令行模式。在命令行输入ipconfig就可以得到触摸板的IP地址。

将LabVIEW VI配置在易失性或者非易失性储存器中

将应用程序配置在嵌入Windows XP式触摸板与嵌入Windows CE式触摸板的步骤是几乎相同的。唯一的区别是在嵌入XP式触摸板中，只可以将应用程序配置在非易失性储存器中，而在嵌入CE式触摸板中，根据选择的目录，可以将应用程序配置在易失性或者非易失性储存器中。不管是运行易失性储存器上的VI还是非易失性储存器上的VI，都必须首先创建一个可执行文件。

给嵌入XP式触摸板创建一个可执行文件

在LabVIEW工程中可以将VI创建成可执行触摸板应用程序。在工程管理器触摸板目标下创建一个生成规范来创建可执行触摸板应用程序。右击生成规范，就可以看到实时程序、源代码发布和压缩包等选项。

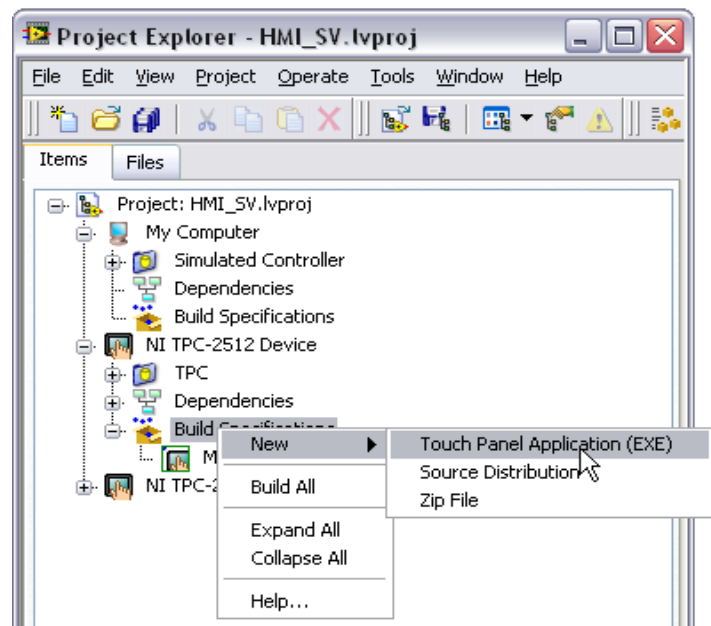


图 8.8 使用LabVIEW 项目床架触摸板应用程序

选择**触摸板应用程序**后，将生成一个对话框。对话框上的信息和源文件选项是生成触摸板应用程序时最常用的两个选项。其他选项在生成触摸板应用程序时很少被使用。

信息选项包括规范名称、可执行文件名以及触摸板目标和本地目的目录。可以修改规范名称和本地目的目录来匹配系统命名和文件格式。通常并不需要改变目标文件名或者目的目录。

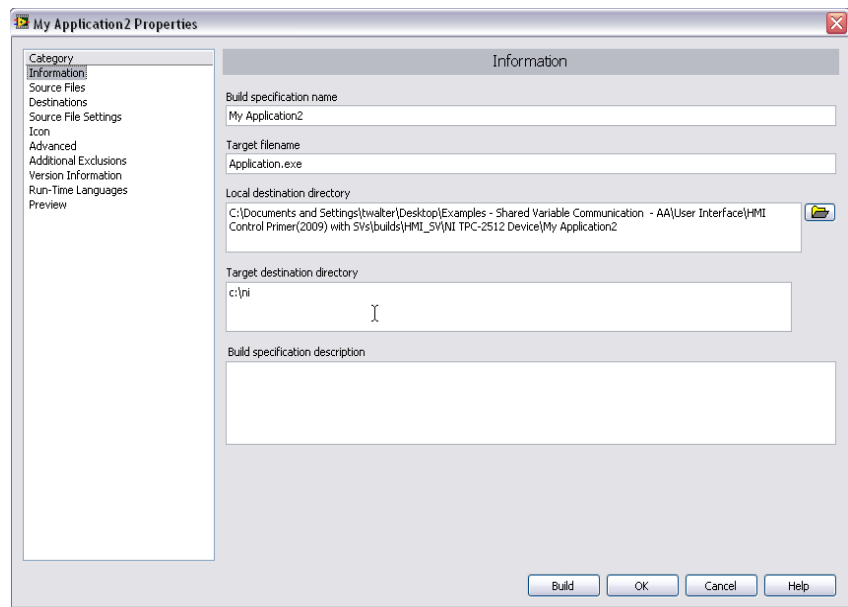


图 8.9 触摸板应用程序属性下的信息选项

源文件选项被用来设置启动VI并包含附加VI和支持文件。从项目文件中选择顶层VI，将其设置成启动VI。对于大多数应用程序只需选择一个VI作为启动VI。不需要包含库文件或设置子VI为启动VI或将这些子VI加入到“始终包含”的文件列表中，除非这些VI是被动态调用的。

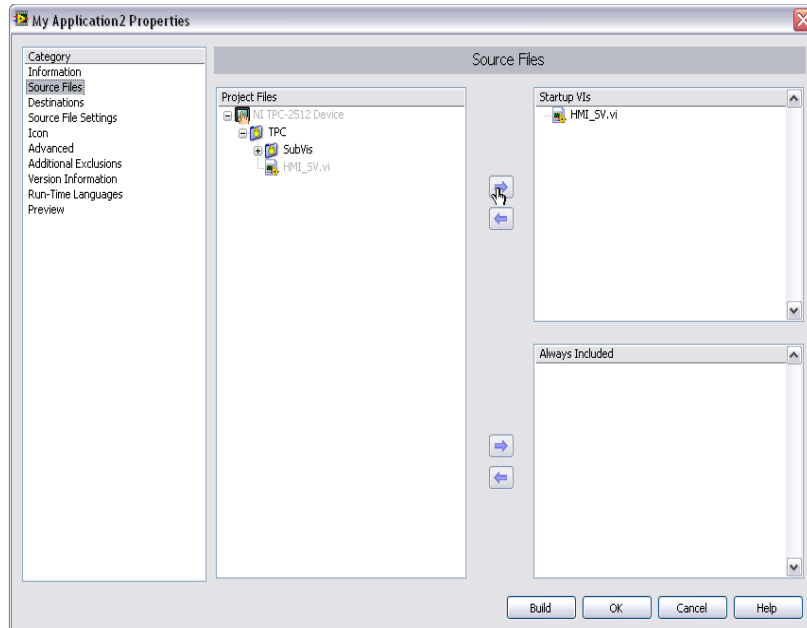


图8.10触摸板应用程序属性上的源文件选项（这个例子中*cRIOEmbeddedDataLogger(主).vi*作为启动*vi*）

设置完必须类别列表中的所有选项后，单击**OK**键来保存生成规范或者直接点击生成键来生成规范。也可以右击保存生成规范选项并选择生成选项来生成应用程序。

生成应用程序后，一个可执行文件就被创建并被保存在开发设备硬件的本地目的目录中。

给嵌入CE式触摸板创建一个可执行文件

在LabVIEW工程中可以从将VI生成触摸板可执行应用程序。在工程管理器触摸板目标下创建一个生成规范来创建触摸板可执行应用程序。右击生成规范，可以选择创建触摸板应用程序、源代码发布和压缩包等。

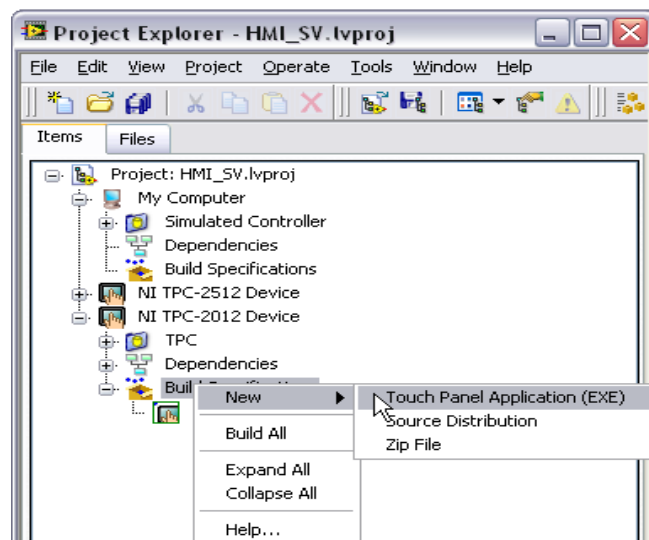


图 8.11 使用LabVIEW 项目床架触摸板应用程序

选择**触摸板应用程序**后，将生成一个对话框。对话框上的应用程序信息、源文件和机器别名选项是生成嵌入Windows CE式触摸板应用程序时最常用的三个选项。其他选项在生成嵌入Windows CE式触摸板应用程序时很少被使用。

应用程序信息选项包括规范名称、可执行文件名以及触摸板目标和本地目的目录。可以修改规范名称和本地目的目录来匹配系统命名和文件格式。通常并不需要改变目标文件名。目标决定了可执行文件是在易失性储存器中还是在非易失性储存器中执行。在嵌入Windows CE式设备中

- \My Documents夹是易失性储存器。如果将可执行文件配置在这个储存器上，在电源关闭后，可执行文件将丢失。
- \HardDisk是非易失性储存器。将“目标应用程序远程路径”设置在\HardDisk\ Documents and Settings下, 则关闭电源，应用程序让然保留在嵌入Windows CE式设备上。

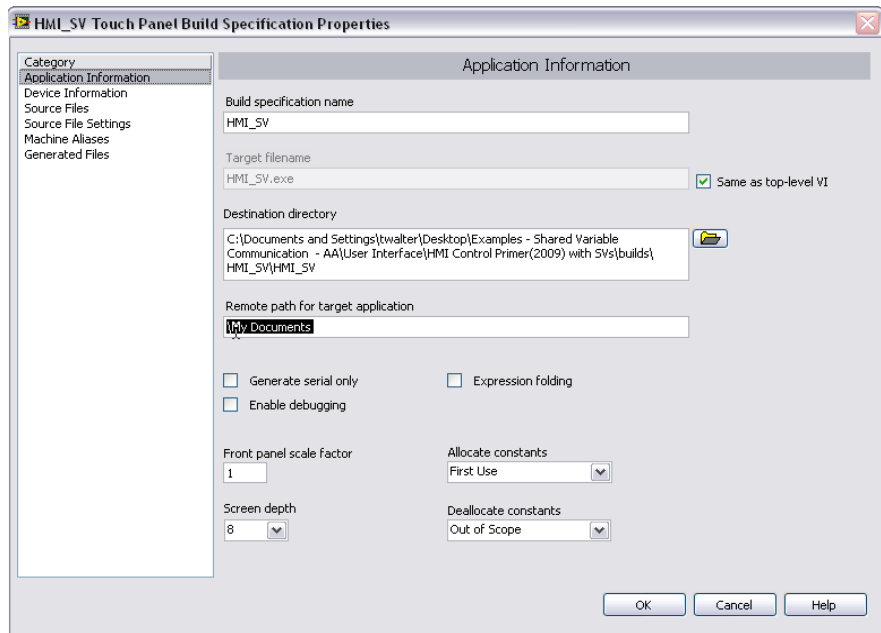


图 8.12 触摸板应用程序属性上的信息选项

源文件选项被用来设置启动VI并包含附加VI和支持文件。从项目文件中选择顶层VI，将其设置成启动VI。对于嵌入Windows CE式触摸板应用程序，只需选择一个VI作为顶层VI。不需要包含库文件或设置子VI为启动VI或将这些子VI加入到“始终包含”的文件列表中。

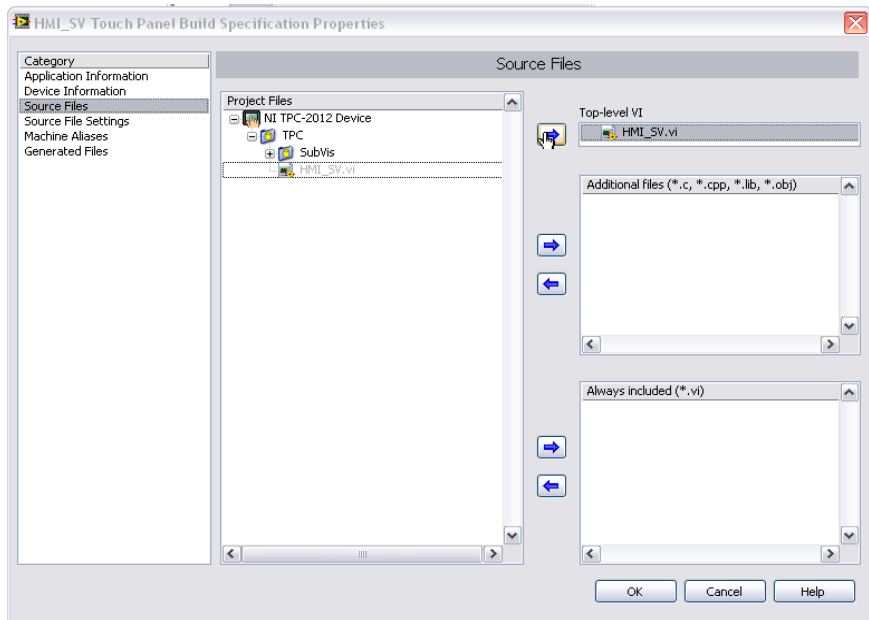


图 8.13 触摸板应用程序 属性上的源文件选项（在这个例子中he HMI_SV.vi被作为顶层VI）

机器别名选项常被用来配置别名文件。如果使用网络发布的共享变量来与其他设备通讯，那么这个选项是必须配置的。一定要检查“配置别名文件”对话框。别名目录包含网络发布的共享变量服务器和IP地址（通常是CompactRIO或者Windows PC的）。更多的关于别名文件和使用网络发布的共享变量配置应用程序的详细信息在这个说明文件的配置部分。

共享变量的三个组成部分来使其在LabVIEW上运行。

网络变量端子

网络变量端子是用来被写入和读取数据的接口简图。每个变量端子都涉及到共享变量引擎的一个网络软件项目。图8.15 展示了一个网络变量的端子、实际网络路径以及工程树状图上相关的项目。

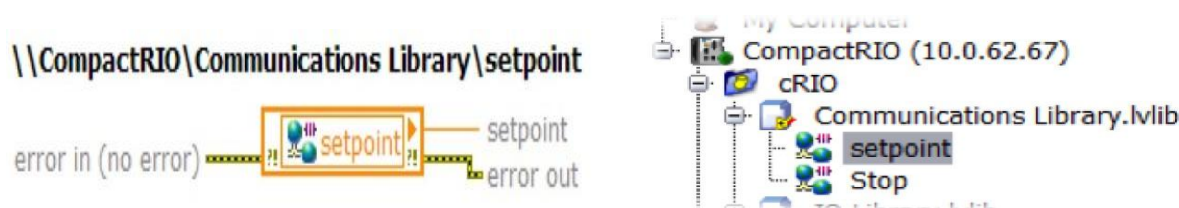


图8.15 网络变量端子及其网络地址

共享变量引擎

共享变量引擎是一种在网上发布数据的软件组件。引擎必须在发布共享变量的实时目标或者计算机上运行。在Windows系统上，共享变量引擎是系统启动上的一项功能。在实时目标上，它是系统启动时加载的一个驱动。

共享变量引擎运行时，它读入保存在非易失性储存器上的数据，并决定哪些变量需要在网上发布。

发布订阅协议

共享变量引擎使用NI公司的NI-PSP（发布订阅协议）来交换数据。NI-PSP是一个使用TCP建立的网络协议，在这个协议里，共享变量客户订阅共享变量引擎上的数据。

向拥有变量的目标上配置共享变量库

CompactRIO系统启动时运行共享变量引擎，引擎进入非易失性储存器访问，来判别变量库是否需要配置。运行一个访问共享变量库的VI或者配置一个访问共享变量库的应用程序时，共享变量库将自动配置。然而，系统可能没有被配置任何变量库。在这种情况下，引擎不能使网络上的任何变量可以使用。

选择以下的两种方法来使共享变量正确得配置在目标设备上。

1. 选择LabVIEW工程上的CompactRIO系统，将变量库放置在设备下并进行配置。这种方法将信息写在了CompactRIO控制器的非易失性储存器中，并使共享变量引擎在网络上创建一个新的数据项目。

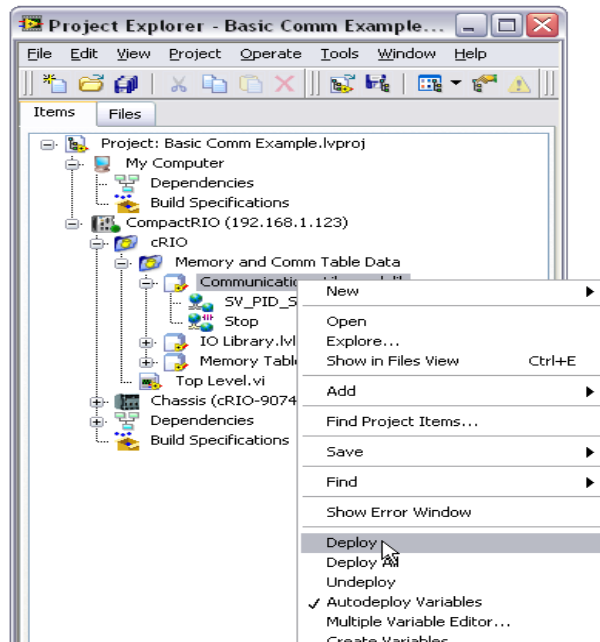


图8.16 选择右击菜单上的配置选项来给实时目标配置变量库

2. 使用应用程序的引用节点可以编程配置运行在Windows上的LabVIEW应用程序的函数库

- 在程序框图上右击，打开函数选项板，进入编程》应用程序控件，将引用节点放置在框图上。
- 使用操作工具，点击方法并选择 库》配置库

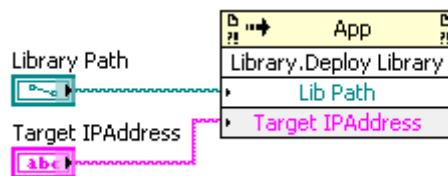


图8.17 使用应用程序的引用节点来正确的配置实时目标上的变量库

- 使用引用节点的配置函数库的路径输入端子来选择包含需要的共享变量的库。使用目标IP地址输入端子来指定实时目标的IP地址。

卸载网络共享数据库

一旦数据库被配置在共享变量引擎，这些设置将一直保存直到人为的将其卸载。

卸载数据库步骤：

1. 启动NI分布式系统管理器（从LabVIEW》工具或者开始菜单）
2. 将实时系统加载到“我的系统”上（操作）给我的系统增加系统）
3. 右击希望被卸载的数据库选择删除

配置共享数据库客户的应用程序

运行只使用共享数据库可执行文件的客户并不需要特别的配置步骤来配置共享库。然后控制器需要将拥有变量的系统名转换成其IP地址。

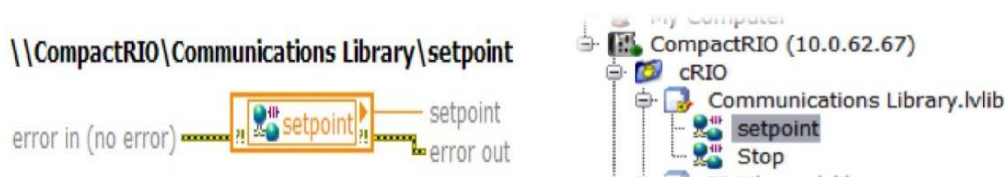


图8.18 网络变量节点及网络地址

为了提供可扩展性，数据信息并不是被硬编在可执行文件里，它是被储存在目标上一个叫别名的文件里。别名文件是一个列出了目标（CompactRIO）逻辑名称及其IP地址（10.0.62.67）的一个可读取的文件。可执行文件运行时，它会读取别名文件并将目标的逻辑

名称转换成IP地址。如果将来更改了配置系统的IP地址，只需要编辑别名文件来重新连接两个设备。对于实时目标和嵌入Windows XP式的目标，它们的生成规范会自动下载这个别名文件。对于嵌入Windows CE式的系统，需要设置生成规范来下载别名文件。

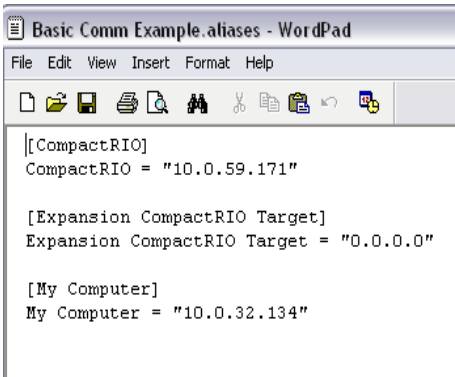


图 8.19 列出目标名称及其地址的可读取的别名文件

如果使用DHCP配置拥有动态地址的系统，就可以使用DNS名来代替IP地址。在LabVIEW工程的目标属性页可以输入DNS名来代替IP地址。

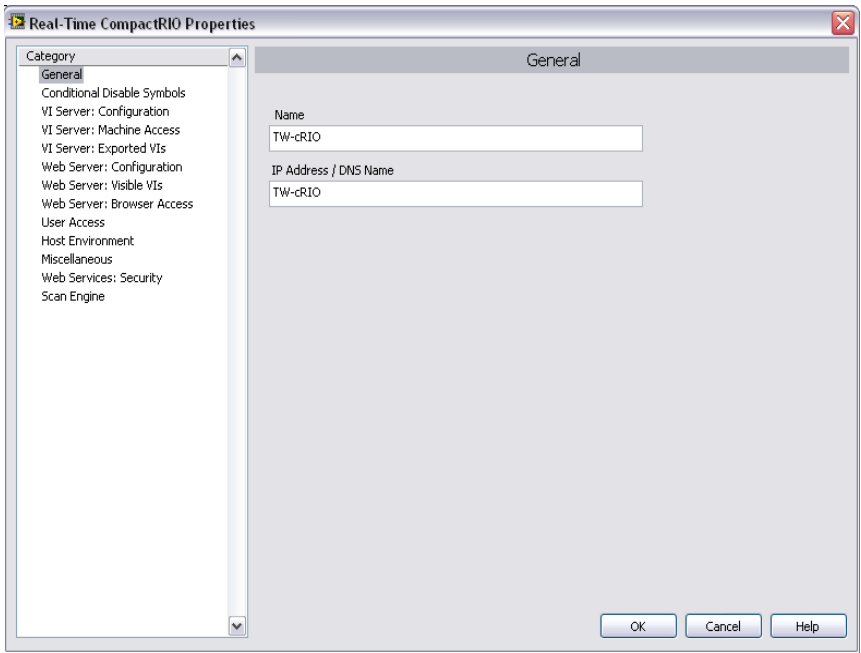


图8.20如果系统使用DHCP，就可以使用DNS名来代替IP地址

如果需要可扩展性，那么开发使用一个通用的目标机器，在应用程序的目标机器名中显示这个目的，这是一个很好的解决方法。作为安装程序的一部分，可以运行一个可执行文件来提示用户“我的电脑”和远程计算机的IP地址或者从其他资源比如数据库来提取这些数据。然后可执行文件能够修改别名文件来体现这个修改。

推荐使用的CompactRIO软件包

NI公司也提供了一些最常用的驱动集合，把这些驱动集合叫做推荐软件包。可以从测量与自动化管理器上把这些软件包安装到CompactRIO控制器上。

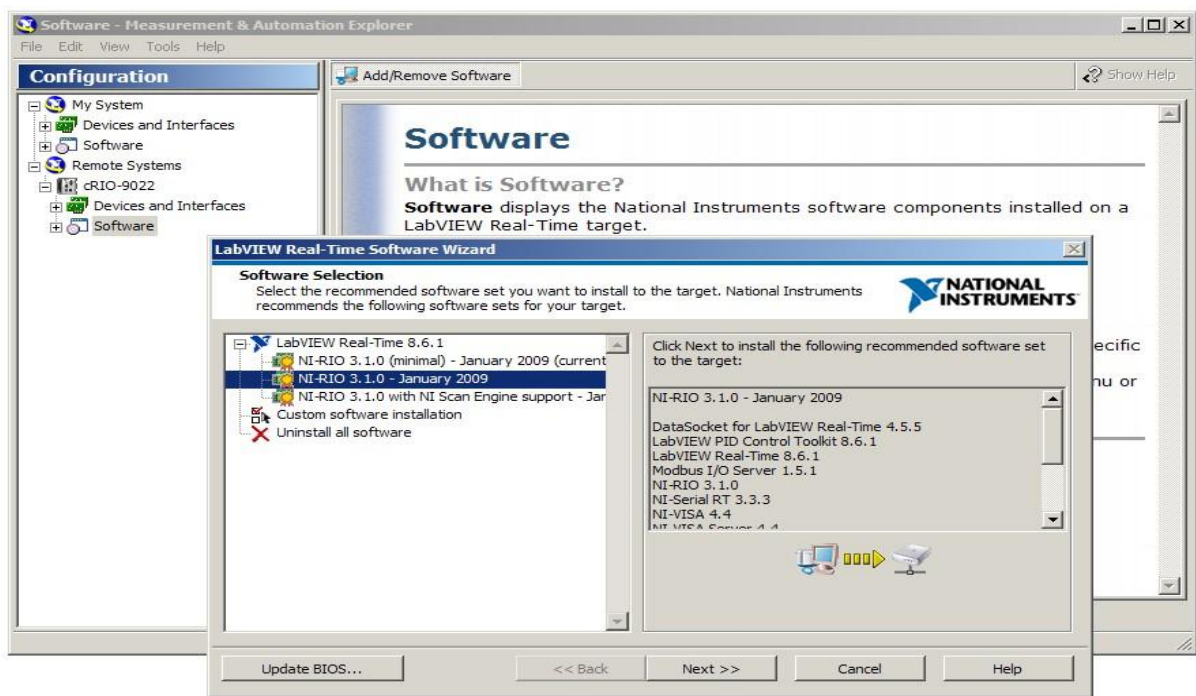


图8.21 推荐软件包被安装在CompactRIO控制器上

推荐软件包保证了每个拥有相同软件集合的实时系统都拥有相同的基础驱动软件包。也就是说既包含了所有必须得软件又使软件包最小化。

系统复制



这部分提供LabVIEW
程序例子

给CompactRIO控制器配置完LabVIEW实时应用程序后，可能也想将这个程序配置到其他相同的实时目标上。可以通过LabVIEW工程和LabVIEW应用程序生成器来按照上述的步骤将已建成的应用程序重新安装到其他目标上。当需要给批量的系统复制和配置已建成的应用程序时这个方法就非常麻烦了。

为了加速这个配置过程，NI公司提供了一系列的系统复制VI来完成LabVIEW实时目标的复制。可以使用这些工具来复制LabVIEW实时控制器硬盘上的所用内容，并将它们拷贝到更多的控制器上。也可以使用这些工具来编程识别在网络上的系统并进行配置。这样就可以不使用测量与自动化管理器和FTP客户端。

具有实时复制功能的VI

自LabVIEW2009开始，安装实时模块后会自带8个具有系统复制功能的VI。

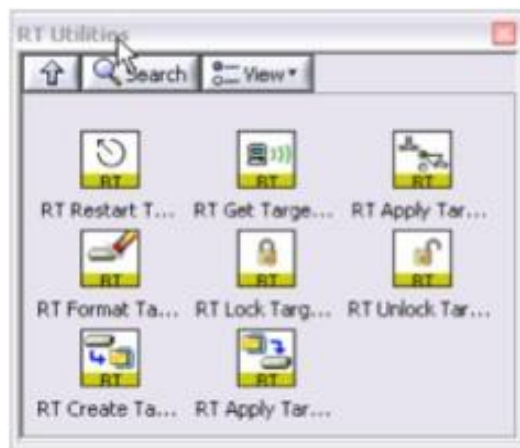


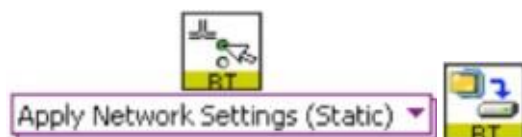
图8.22 具有实时功能的VI，可以实现控制器的自动配置



使用RT Get Target Information VI，可以搜集到网络上的实时设备。这个VI通过IP地址或者MAC地址来找到这些设备，还能够找到在子网上注册的所有NI实时设备。



一旦RT Get Target Information VI放置到网络上的可编程的目标后，RT Create Target Disk Image就会连接到目标，并将目标的所有内容通过FTP转移到电脑上。这些内容被保存到计算机的一个压缩文件里。这个复制过程会持续几分钟。



可以使用RT Apply Network Settings和RT Apply Target Disk Image VI来配置新的控制器。第一个VI设置目标的网络环境，第二个VI将取走以前保存的磁盘镜像并将全部内容下载到新的实时控制器上。这个工程需要几分钟。

这些VI也可以使用密码将FTP服务器锁定在实时控制器上，或者将其上的FTP服务器解锁。

创建复制功能

通过这些VI，可以运行在Windows的LabVIEW上创建自定义复制功能。

在这个向导里有一个使用了这些VI的预建的复制功能的例子。运行桌面上的应用程序时，它提供了复制和管理控制器和镜像的图示使用接口。

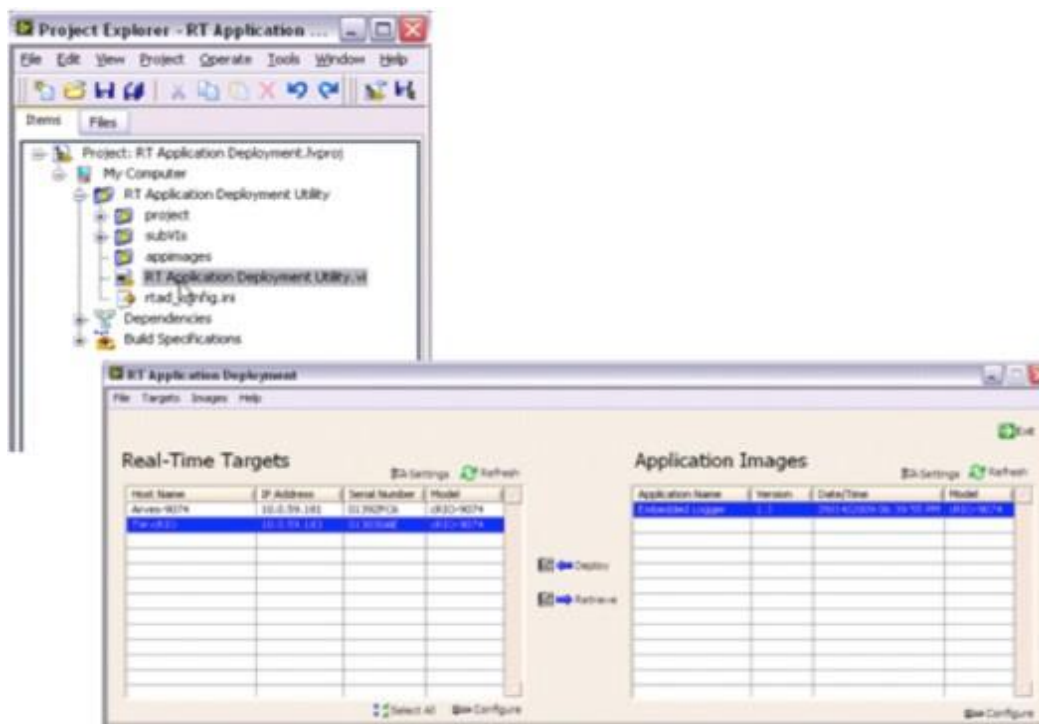


图8.23 像这个复制应用程序一样，也可以创建一个高级的复制程序

这个应用程序可以自动为实时目标扫描网络，也可以为保存的应用程序镜像扫描实时目标目录。可以检索到目标磁盘镜像并将其保存在计算机上并将应用程序镜像安装在目标上。这个过程需要持续几分钟。可以使用LabVIEW工程下readme.doc文件里的这个例子来获得更多的信息。

NI-RIO系统复制工具

通过提供LabVIEW API，NI-RIO系统复制工具给实时复制工具提供了额外的支持。这些API可以编程下载FPGA bitfile文件来刷新后面板上的存储器。如果配置了需要立即启动运行FPGA代码的应用程序，而这些代码不能通过标准的方法从实时程序上下载得到，那么NI-RIO系统复制工具就显得非常有用。使用这些工具，可以删除和下载FPGA bitfile文件来刷新内存，以及怎么从刷新的内存中下载VI。只有安装NI-RIO之后这些工具才能使用。



Set RIO Device Settings.vi

从刷新的内存上下载bitfile文件时，使用Set RIO Device Settings.vi配置。



Download Bitfile.vi

这个VI向一个或者多个FPGA目标上下载特定的bitfile文件或者删除已存在的bitfile文件。输入端子是主机的IP地址、FPGA目标源（RIO0）、bitfile文件路径和执行操作（下载或者删除bitfile文件）。

知识产权保护

知识产权涉及到了任何由个人或者公司独立开发的特有的软件或者应用程序算法。所涉及的对象可能是一个特殊的控制算法也可能是一个完整的应用程序。知识产权通常需要花费大量的时间去申请，它给公司提供了一个与其他竞争产品区别的方法。因此保护软件的知识产权就非常重要。LabVIEW开发工具和CompactRIO提供了保护和锁定知识产权的功能。一般来说，可以使用两种方法来保护知识产权：

锁定算法或者程序代码以防止被复制或者修改

如果给一个特殊的功能创建了一个算法，比如执行一个高级的控制函数、执行自定义过滤等，且想把这个算法以子VI的形式发布出去，

但是不允许他人看到或者修改实际的算法。就可以使用知识产权保护着降低支持级别来组织其他人修改或者破坏算法。

将程序代码锁定在特殊的硬件上以防止被复制

如果想确保竞争者不能复制在其他的CompactRIO上运行的系统，或者想要客户返回寻求服务或者支持。

锁定算法或者程序防止被复制或修改

保护已配置的程序

通过默认锁定和不能被打开，LabVIEW能够保护所有的已配置程序或者CompactRIO上的所有开启后运行的应用程序。不像其他现成产品或者将源代码储存在控制器上只通过密码保护的可编程式逻辑控制器，CompactRIO不需要将源代码储存在控制器上。

运行在实时处理器上的程序被编译成一个可执行文件，但不能被“反编译”成LabVIEW程序。同样运行在FPGA上的程序被编译成一个bitfile文件，但不能被“反编译”成LabVIEW程序。为了将来方便调试和维护，也可能将LabVIEW工程储存在控制器上或者从运行的代码中调用原来的VI，但是通过默认设置，任何被配置在实时控制器上的代码都被保护起来防止复制或者修改源代码。

保护特殊的VI

有时候会将LabVIEW源代码提供给最终客户，来执行专用化定制或者维修，但仍想保护特殊的算法。LabVIEW提供了一些机制，这些机制在提供可用于VI的同时保护他们的知识产权。

方法1：通过密码保护LabVIEW程序代码

密码保护使得用户在编辑或者查看一些特殊VI的程序框图时，必须输入密码。所以将VI提供给他人同时密码保护能够保护源代码。密码保护阻止那些没有密码的人来编辑或查看LabVIEW子VI。然而如果忘记了密码，也就不能再解锁VI了。因此强烈建议将一个没有密码保护的备用文件保存在一个安全的地方。

进入文件》VI属性，选择保护类别。可以看到三个选项：开启（默认的VI状态）、锁定（没有密码）和密码保护。单击密码保护，弹出一个窗口，在窗口上输入要设置的密码。这样VI就被设置成密码保护了，在下次启动LabVIEW的时候密码保护就开始起作用。



图8.24密码保护LabVIEW程序

破解LabVIEW密码保护机制是非常困难的，但是没有任何密码算法是100%安全的。如果需要绝对确保其他人员不能接触到源代码，那么应该考虑移除程序框图。

方法2：移动程序框图

完全移除程序框图可以确保VI不能被修改或者打开。很像一个可执行文件，被发布的程序不再包含源代码。因为程序框图不能重新创建，所以在使用这项技术的时候确保保存了一个备用文件。创建一个源发布时可以选择移除程序框图。源发布就是将一个文件集合打包发送给开发者使用。可以为一些特殊的VI配置密码保护或者移除程序框图，或者应用其他设置。

完成以下步骤来建立源发布。

1. 在LabVIEW工程中右击生成规范，在弹出的快捷菜单中选择新建》源发布，弹出源发布属性对话框，把VI增加到发布。
2. 在源发布属性对话框的源文件设置页，取消使用默认保存设置，并选择移除程序框图，来确保LabVI移除了程序框图。
3. 创建源发布来生成一个没有程序框图的VI副本

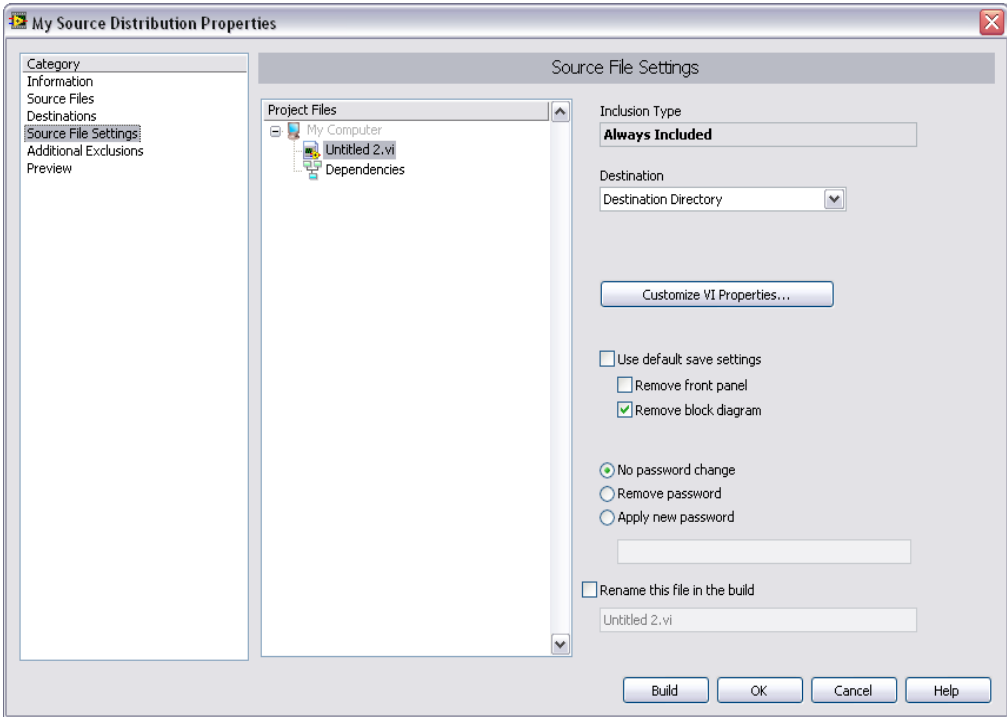


图8.25 移除LabVIEW VI的程序框图

注意： 如果保存没有程序框图的VI，确保没有覆盖原来的VI。将这些VI保存在不同的路径或者使用不同的名字。

将程序代码锁定在硬件上以防止知识产权被复制

一些原型设备制造商和机器制造商将安装程序锁定在一个特定的系统里保护他们的知识产权。使CompactRIO控制器上的安装程序不被锁定在硬件上，可以很容易移动并在其他控制器上执行成为默认设置，就使得系统容易复制。阻止客户或者竞争者复制系统的一条有效途径就是将程序代码锁定在系统硬件的一些特定区域里。这个办法保证了客户不能从买到的CompactRIO中拷贝程序代码在其他的CompactRIO上运行。可以将应用程序代码锁定在CompactRIO系统硬件的不同组件里。这些组件包括：

- 实时控制器的Mac地址
- 实时控制器的序列号
- CompactRIO后面板的序列号
- 特殊模块的序列号
- 第三方串行加密钥匙

通过下面的步骤可以将任何应用程序编程锁定到上面提到的任何一个硬件里，防止使用者复制程序代码。

1. 得到设备的硬件信息。涉及到下面程序的更多信息可以编程得到。
2. 使用比较函数选项板上的Equal? 函数来比较获得的数值与预设的应用程序设计值。
3. 将比较结果连接到Case结构的输入端子

4. 将应用程序放置到正确的分支上，错误的分支设置成空的。

执行这些步骤以确保应用程序不能被复制或者在其他CompactRIO硬件上不能使用。

许可证

配置许多硬编码的系统时，硬件的识别可能不是很理想，因为这需要给每个安装系统再编译以及手工改变得到源代码。通过使用保存在CompactRIO控制器上与应用程序区分开的许可证可以解决这个问题。每个系统的许可证文件可以很容易的更新，而不用修改应用程序。除了读取MAC码及序列号，VI可以打开许可证文件来检查许可证是不是合法的。为了安全，每个安装系统的许可证都是特定的，输入的许可证代码必须和硬件的特征精确吻合，比如MAC地址。可以从Windows系统和实时操作系统上编程得到MAC和序列号，所以可以为系统安装开发一个LabVIEW应用程序，这个程序可以自动识别CompactRIO系统并产生和安装正确的许可证文件。

得到CompactRIO系统的MAC地址



这部分提供

LabVIEW例子的代码

可以手工的从Windows命令行或者控制器的网络设置列表上的MAX来得到实时控制器的MAC地址。也可以编程得到控制器的MAC地址。

从Windows上获得MAC地址

执行以下步骤，从Windows编程得到MAC地址

1. 运行Real-Time»Real-Time功能板上的RT Ping Controllers VI。这个VI返回子网上的所有实时控制器的网络信息。

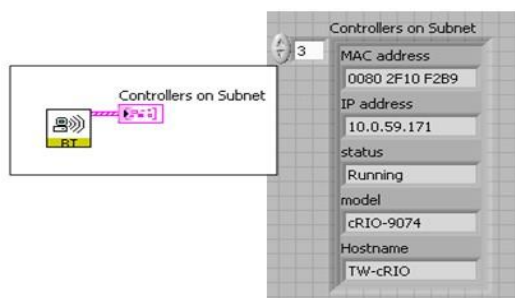


图8.26 从Windows上获得网络实时目标的MAC地址

2. 通过IP地址或者主机名从RT Ping Controllers VI返回的数据中搜寻合适的实时控制器
3. 这个簇返回控制器的MAC地址，被RT Ping Controllers VI 返回的MAC地址是一组十六进制的字符串。

从实时目标上获得MAC地址

执行以下步骤，从实时控制器上编程获得MAC地址

1. 运行Real-Time»Real-Time功能板上的RT Ping Controllers VI。将一个内容为错误和当地主机的簇连线到系统位置输入 端子。程序运行时这个VI就返回实时控制器的信息。

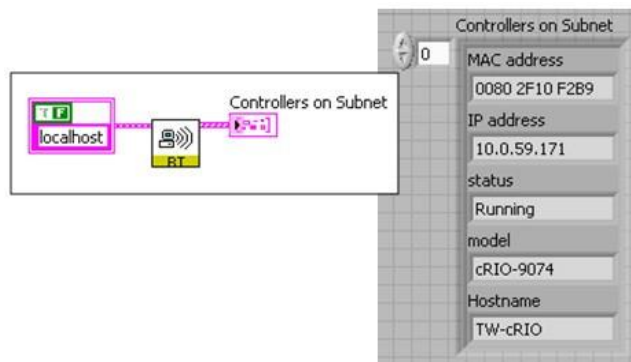


图8.27 在实时目标上获得MAC地址

2. 在RT Ping Controllers VI返回的数据中搜寻控制器的MAC地址。被RT Ping Controllers VI 返回的MAC地址是一组十六进制的字符串。

CompactRIO信息检索工具

使用CompactRIO信息检索工具可以得到CompactRIO系统的其他信息比如序列号。这些VI可以运行在Windows或实时控制器上，可以获得关于当地或远程CompactRIO控制器信息、后面板信息和模块的信息包括类型和三个系统组件各自的序列号。

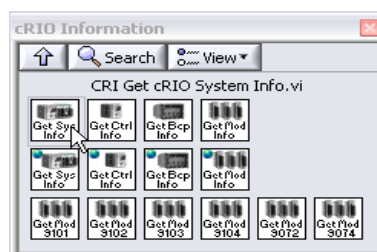


图8.28 CompactRIO信息检索工具 返回如序列号等信息

如果使用LabVIEW实时控制系统复制工具，也可以编程检索到实时目标的序列号。

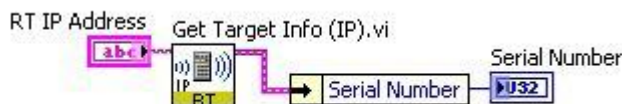


图8.29 使用LabVIEW实时控制系统复制工具得到序列号

连接其他平台

本章致力于通过使用CompactRIO系统创建嵌入式控制系统的体系结构。相同的基础技术和结构也可以在NI公司的其他控制平台包括PXI和NI Single-Board RIO上使用。因此可以将这些算法和结构应用于其他需要不同硬件的项目，或者只是简单的将应用程序移植在不同的平台间移植。然而CompactRIO拥有一些易于学习和加速开发的特征，不是所有的设备都拥有这些特征。这节是关于在不同的平台间移动应用程序时需要考虑的事项以及怎么将应用程序连接到NI Single-Board RIO。

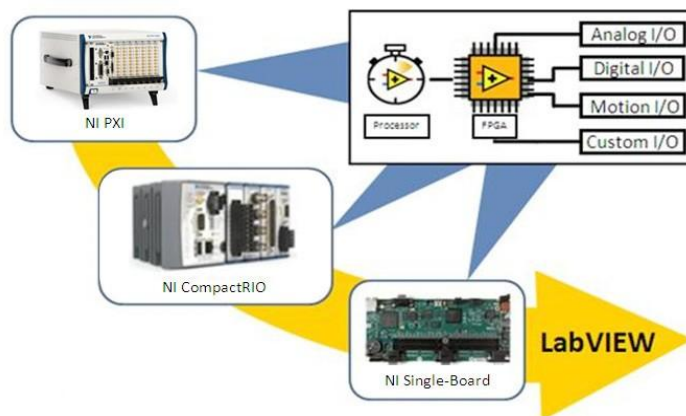


图8.30 通过LabVIEW可以在高性能的PXI、低级的NI Single-Board RIO 以及CompactRIO之间使用相同的技术

LabVIEW程序代码的可移植性

LabVIEW是一种交叉平台的编程语言，它可以给多种处理器结构和操作系统编译。大多数情况都是在LabVIEW上编写算法，然后将其移植到其他LabVIEW目标上。实际上，甚至可以将LabVIEW程序编译给任何一个允许将LabVIEW程序连接到自定义硬件上的32位处理器，这个处理器允许将LabVIEW程序代码连接到指定的硬件上。

连接不同的平台时，最需要的改变就是硬件物理I/O的改变。

NI Single-Board RIO(嵌入式单板平台)

NI Single-Board RIO是为应用程序设计的只有一个板卡的CompactRIO版本。从根本上来讲它也是一个不同的设计产品，它使用处理器和FPGA并且可以容纳3个C系列的模块。NI Single-Board RIO与CompactRIO不同之处在于它直接将I/O建立在板卡上。NI Single-Board RIO的特征在于依据所使用的模块，它可以拥有110个3.3V的双向数字I/O线、高达32个模拟输入、4个模拟输出、32个24V的数据输入与输出线。

LabVIEW FPGA 编程

NI Single-Board RIO目前不支持浏览模式。而是需要编写一个LabVIEW程序来读取FPGA上I/O的数据并将其嵌入到储存器列表中。这部分介绍一个有效的单点I/O通讯的FPGA结构，这个结构与下节中的浏览模式非常相像，并展示怎么来转换一个使用浏览模式的应用程序。

板载I/O和I/O模块

根据应用程序对I/O的要求，在创建整个应用程序时可能只使用NI Single-Board RIO的板载I/O，或者需要增加一些模块。在可能的情况下，尽量使用NI Single-Board RIO板上可用的I/O模块。NI Single-Board RIO板上可用I/O以及相应的模块如下：

- 110个通用的3.3V（5V容差、与晶体管-晶体管逻辑电路协调）数字I/O（没有相应的的模块）。
- 32个单端/16个不同通道、16位模拟输入，总计达250Ks/s(NI9205)。
- 4个通道、16位模拟输入，同时100KS/s(NI9425)。
- 32个通道、24V沉没数字输入（NI9425）
- 32个通道、42V数字源输出（NI9476）

NI Single-Board RIO可以容纳高达3个附加的C系列模块。所以它不能运行那些需要3个以上附加I/O模块的应用程序。这时需要考虑将CompactRIO整体系统作为安装目标。

FPGA大小

NI Single-Board RIO上最大的可用的FPGA是Xilinx 2M system gate Spartan-3 FPGA。CompactRIO目标提供使用Spartan-3FPGA和更高、更快的Virtex-5 FPGA的版本。可以将一个目标增加到LabVIEW工程里来检测程序代码是否安装到硬件上了。同样可以通过为一个仿真RIO设备编译FPGA程序周期性检查应用程序来开发自己的应用程序。这样就可以很好的理解FPGA应用程序将能够多大程度的安装到Spartan-3FPGA。

将CompactRIO应用程序连接到NI Single-Board RIO或者R系列设备

执行以下4个主要步骤就可以将CompactRIO应用程序连接到NI Single-Board RIO或者PXI/PC R系列I/O设备。

1. 创建一个NI Single-Board RIO或者R Series工程和相关的I/O通道
2. 如果使用CompactRIO扫描模式，就创建一个基于FPGA的LabVIEW扫描应用程序接口。
 - 创建一个LabVIEW FPGA I/O扫描（模拟输入、模拟输出、数字I/O及专业的数字I/O）。
 - 将一个I/O变量别名转换成单进程的共享变量，并使共享变量能够实时的先进先出。
 - 创建一个带有缩放和基于当前变量值列表的共享变量的实时I/O扫描。
3. 给新的设备编译LabVIEWFPGA。
4. 测试和验证更新的实时和FPGA程序。

连接来自CompactRIO的应用程序和NI Single-Board RIO或者R Series FPGA设备的第一步就是寻找目标平台上对应的I/O类型。对于那些不能与NI Single-Board RIO或者R Series的板载I/O相连的I/O，需要增加C系列模块。CompactRIO上的所有C系列模块都可以与NI Single-Board RIO和R Series相容。必须使用NI9151 R 系列插槽扩展器来给R系列数据采集设备增加C系列I/O。

如果被连接的应用程序最初是使用扫描模式编写的，那第二步才是必要的。下面的例子将展示怎么把应用程序的扫描模式部分换成NI Single-Board RIO和R Series支持的I/O方法。

如果被转移到NI Single-Board RIO上的应用程序不使用扫描模式，那么连接就几乎完成了。跳过第二步，将实时和FPGA源代码增加到新的NI Single-Board RIO工程上，编译FPGA VI。到现在就可以运行和验证应用程序的功能了。因为ComactRIO和NI Single-Board RIO都是基于RIO结构和可重复使用的C系列I/O模块设计的，所以连接这两个应用程序是非常简单的。

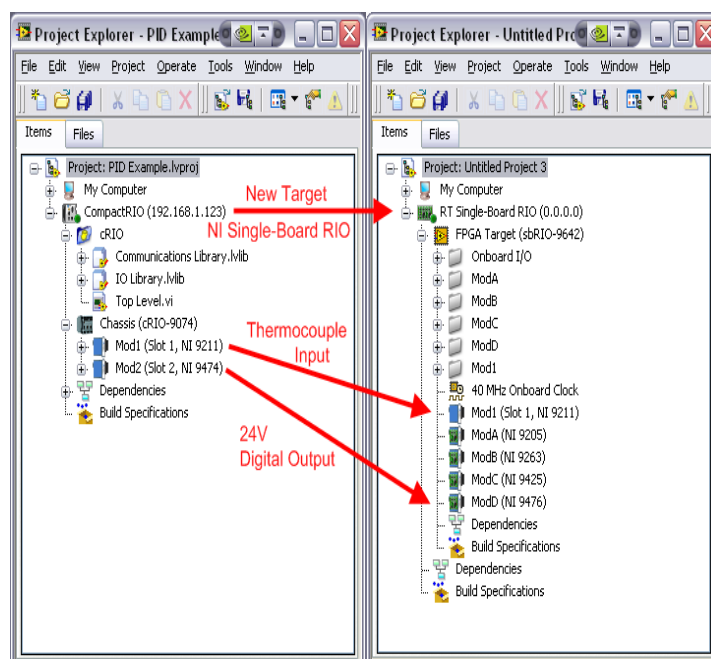


图8.31 连接来自CompactRIO应用程序与另一个目标的地比不就是在这个目标上寻找替换的I/O



这节提供LabVIEW

例子的程序代码

如果在原始应用程序中使用扫描模式，就需要创建一个简单的扫描模式形式的FPGA，这时因为NI Single-Board RIO和R Series设备都不支持扫描模式。在FPGA中创建扫描引擎与在“使用LabVIEW FPGA编程”一节中讨论的将FPGA单点数据插入到实时扫描中非常相似。将扫描模式替换成相似的基于扫描引擎的FPGA和当前数值表共有三步。

1. 创建一个LabVIEW FPGA I/O扫描引擎。
2. 将IOV替换成单进程共享变量。
3. 将FPGA数据插入到LabVIEW实时模块中基于当前数值表的共享变量中。

第一，创建一个LabVIEW FPGA VI，这个VI以扫描引擎里设置的特定速率采集和更新模拟输入和输出通道里的数据。可以使用IP块重新创建数字功能项，比如计数器、脉宽调制器和正交编码器。

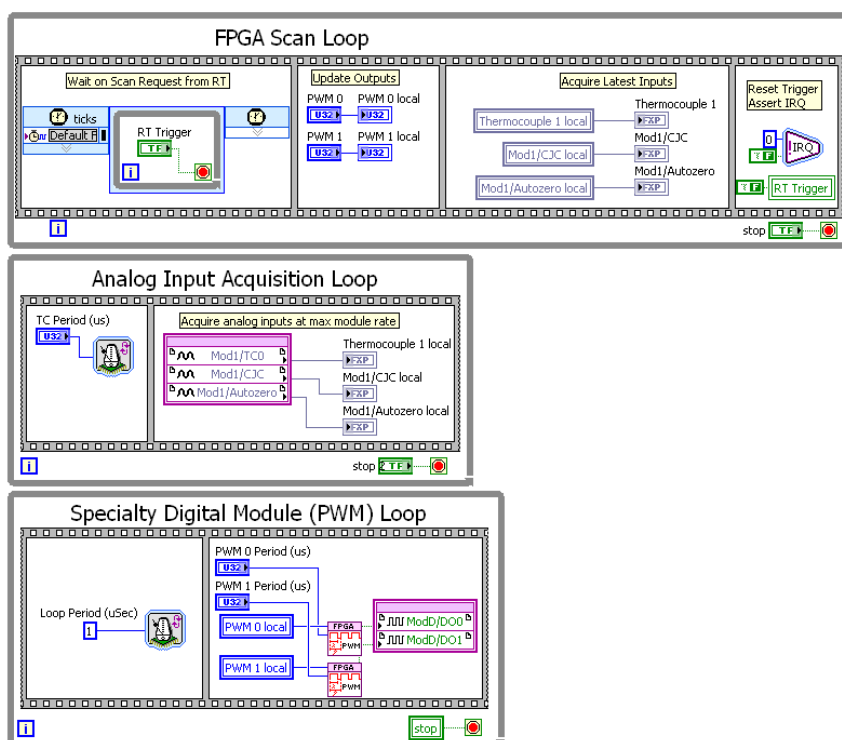


图8.32 将一个简单的FPGA应用程序开发成一个FPGA扫描引擎

在FPGA里执行一个简单的扫描引擎后，就应该将实时应用程序部分与自定义FPGA扫描而不是扫描模式里的当前数值表联系起来。为了完成这个过程，首先需要将所有的I/O变量别名转换成单进程共享变量同时使实时变量能够先进先出。这两个变量的主要区别在于I/O能够自动被驱动器更新来体现输入或者输出通道的状态，而单进程变量不能自动被驱动器更新。进入每个IOV别名的属性页可以改变他们的类别，将其改变成单进程。

建议： 通过将变量输出为文本编辑并改变他们的属性，可以轻松的将一个IOV别名库转换成共享变量，这样就可以转换批量的变量。首先创建一个允许单个元素实时的先进先出的单进程共享变量“模板”，并将变量库输出为电子表格编辑，这样就可以保证很容易找到正确的属性。在电子表格编辑器里，删除IOVs专用列，并将这些专用数据保存在IOV行里的共享变量里。然后将修改的变量库输入到新的工程里。IOV别名被以单进程共享变量的形式输入进来。因为LabVIEW根据库名和变量名来访问这些IOV别名和共享变量，所以VI里的所有IOV别名实例都会自动更新。最后删除在转移过程中创建的共享变量模板。

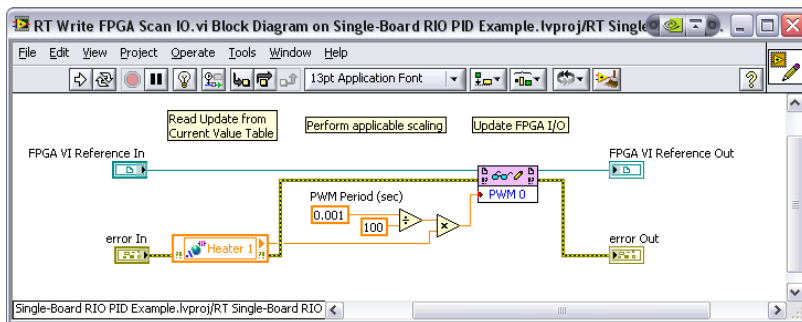


图 8.35 RT Write FPGA Scan I/O VI 使用实时的先进先出单进程共享变量从存储列表中得到数据，然后适当的缩放转化给FPGA Scan VI，并将这些数值传递给FPGA VI

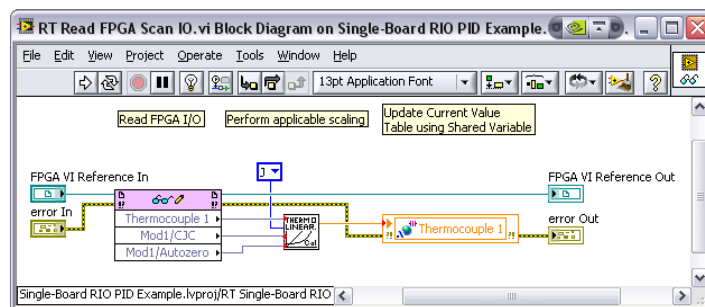


图 8.36 RT Write FPGA Scan I/O VI 从FPGA扫描中得到所有的更新，将其通过适当的转化和缩放通过实时的先进先出单进程共享变量传递给存储表
创建完自定义FPGA I/O扫描的主接口来代替扫描模式，这样就可以在新目标上测试和运行链接的应用程序。确保FPGA VI已被编译且工程里的实时和FPGA目正确得配置了有效的IP地址和RIO源名称。FPGA VI被编译完后，就可以连接实时目标并运行应用程序。

NI Single-Board RIO, CompactRIO, and R Series FPGA I/O设备上使用了共同的RIO结构，所以在这些设备上编写的LabVIEW程序就很容易连接到其他设备上。就像这节中提到的一样，通过正确的配置，不需要修改程序就可以在新的目标上移植应用程序。当使用平台上的专有特性时，比如CompactRIO扫描模式，连接过程可能有些复杂。但在这样情况下，只需要为移植修改程序的I/O部分。在这些情况下，所有的LabVIEW程序和控制算法通过RIO硬件平台完全可以连接和重复使用。

附录 A

CompactRIO 入门指南

CompactRIO 入门教程

这篇教程向你展示了如何获得你的CompactRIO系统并创建第一个项目。它介绍了硬件配置、必要的软件包安装、以及用于配置CompactRIO系统的简单项目开发。

CompactRIO 及其组件

一个CompactRIO系统由以下五个主要部分组成：

- 1. 机箱
- 2. 控制器（可能是集成与设备中的控制器如cRIO-907x）
- 3. 模块
- 4. 系统附件
- 5. 软件

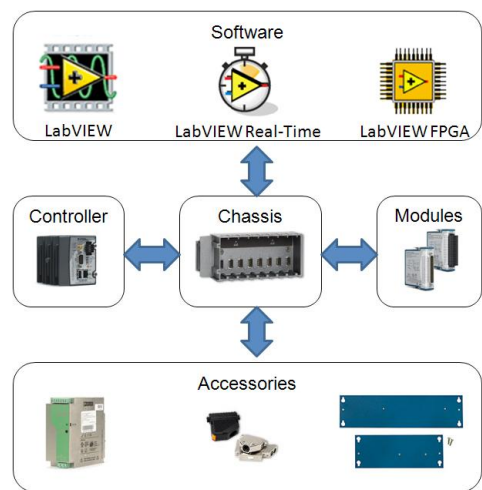


图9.1 完整的CompactRIO系统组件

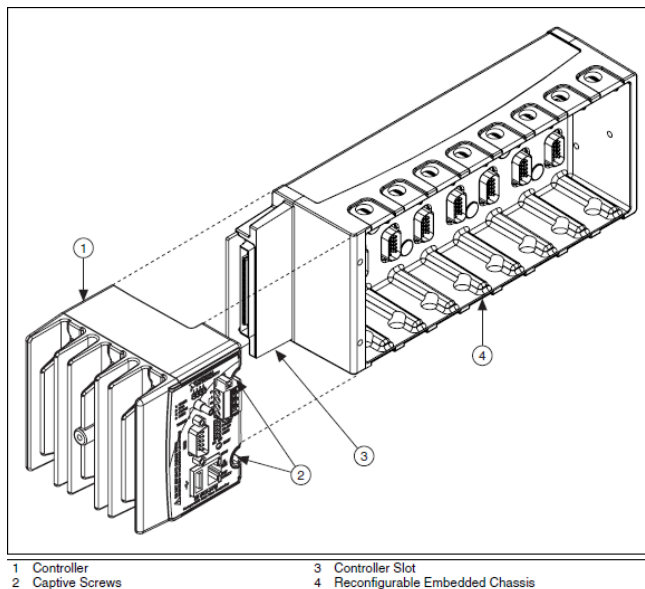
硬件安装与配置

将CompactRIO系统组件从盒中取出后，采取以下步骤为你的系统进行软件的安装与编程。

连接控制器、机箱、模块

如果你拥有的是一个模块化控制器，将它与可重构的FPGA机箱组装在一起。cRIO-907x系列将机箱与控制器集成在一起，因此不需要组装。将控制器连接至以底盘，须遵循以下步骤：

- 确保在断开电源的情况下完成组装，保证控制器与机箱插口对应。
- 将控制器与机箱的控制器插槽对准，沿着滑道用力将其插牢。用2号飞利浦螺丝刀上紧两个固定在控制器前面的螺钉，如图



9.2所示。

图 9.2 控制器与机箱的连接

控制器电源搭接

为系统搭接一个适当的直流电源供电，如24v供电。将电源正极连接至V1终端，负极连接到两个C终端中的任意一个。一些控制器配有备用电源接口，可以将备用电源连接至V2终端。

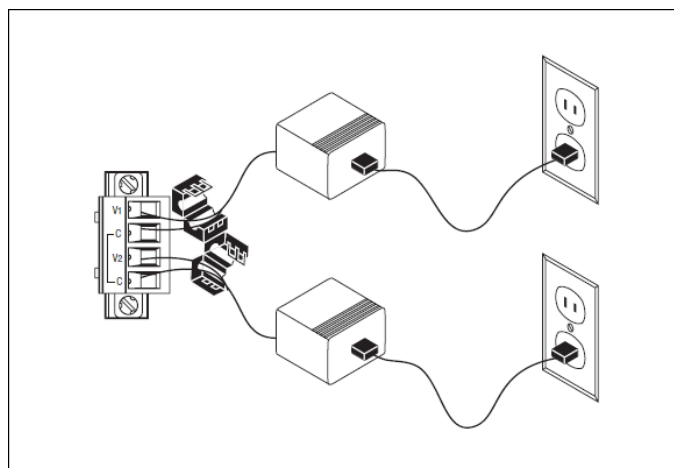


图 9.3 CompactRIO控制器电源搭接

指拨开关配置

CompactRIO控制器配置了五个指拨开关-安全模式、控制台、IP重置、无应用程序、用户1。National Instruments仪器出厂时，所有开关被设定在OFF位置。配置控制器时，开关应该任然停留在OFF位置。

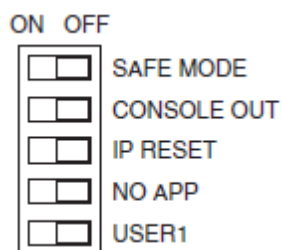


图9.4 配置你的CompactRIO控制器时关闭开关

指拨开关及其使用方法在[CompactRIO操作说明和技术规格手册](#)中单独介绍。

连接CompactRIO系统至你的PC

你有两种方法将CompactRIO系统连接至你的PC：

1. 通过路由器或交换机，将系统与PC连接在同一子网段下。
2. 通过交叉网线，将CompactRIO系统直接连接至你的PC。

在MAX中配置你的CompactRIO系统

所有硬件连接完成后，你可以通过Measurement & Automation Explorer软件来配置你的CompactRIO系统（**开始»程序»National Instruments»Measurement & Automation Explorer**），这样才可以在LabVIEW程序中定位和编程。在使用MAX配置你的CompactRIO系统之前，你的主机电脑需要预先安装LabVIEW Real-Time模块、NI-VISA和NI-RIO软件。NI-RIO程序需要在最后安装。

打开CompactRIO系统电源，在MAX程序中拓展打开远程系统，CompactRIO系统名称最初默认显示为cRIO-[控制器型号]。你可以在识别设置中修改名称。你的系统以树形方式拓展开后，右侧窗口详细显示CompactRIO系统的IP设置。IP默认设置为0.0.0.0。

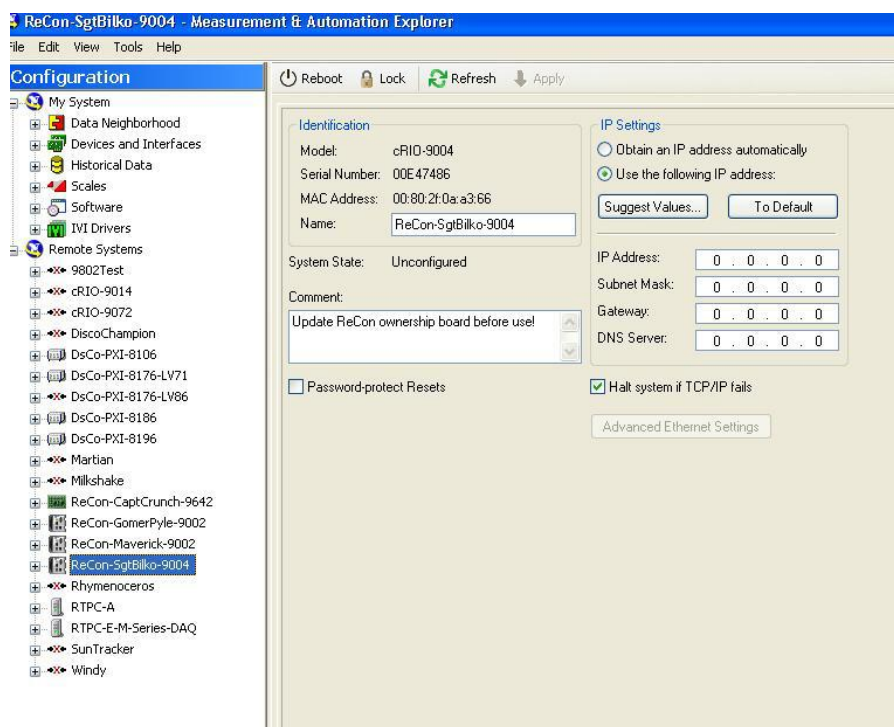


图 9.5 IP 默认设置为0.0.0.0。

如果你是通过支持DHCP的路由器连接你的CompactRIO系统，你可以将系统设置为自动获取IP地址。如果你是通过网线连接你的CompactRIO系统，必须将你的主机电脑设置为静态IP地址。互联网地址编码分配机构（IANA）已将下面三个IP地址段授权给私人网络使用。

- 10.0.0.0 – 10.255.255.255
- 172.16.0.0 – 172.31.255.255
- 192.168.0.0 – 192.168.255.255
- 169.254.0.0 – 169.254.255.255（使用网线的链路本地地址）

私人网络中，每个机器必须分配一个唯一的IP地址。例如，可以将192.168.0.1分配到主机，192.168.0.3分配到远程系统。必须先将

主机和CompactRIO系统配置在同一子网下，才能配置子网掩码。在命令提示符中键入**ipconfig**，来查看主机的子网掩码：**开始»运行»cmd**

主机系统TCP/IP设置，需要完成以下步骤

- 1. 打开网络连接（可从控制面板进入）
- 2. 右键单击图标选择属性。在一般窗口中，点击**Internet协议（TCP/IP）**，然后点击**属性**。
- 3. 指定一个IP地址，点击“使用下面的IP地址”并输入IP地址和子网掩码。

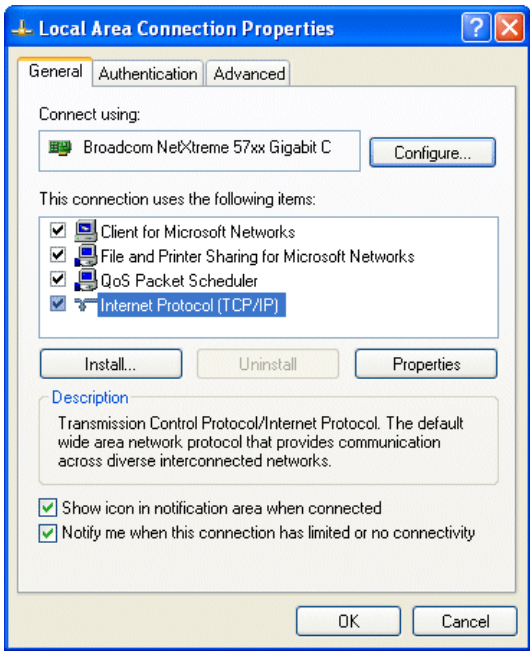


图 9. 6主机系统TCP/IP设置

MAX中的配置步骤

在MAX中，按照以下步骤完成你的CompactRIO系统配置：

- 1. 在**IP设置窗口**中设置IP地址：
 - 如果使用支持DHCP的路由器连接，选择**自动获取IP地址**。
 - 如果使用网线连接，选择**使用下面的IP地址**并为CompactRIO系统指派一个与PC在同一网段的IP地址。
- 2. 可以在识别窗口中的**名称**一栏中修改CompactRIO名称，点击应用。如果网关一栏没有填写IP地址，程序会提示你的CompactRIO系统与网关处于不同的子网中。点击确定忽略这侧提示。如果在网关一栏中输入地址，确保它与默认网关值相对应，并且不能是0. 0. 0. 0. 其会导致与MAX联系中断。在命令提示符中键入**ipconfig/all**，来查看网络的默认网关。
- 3. 为CompactRIO系统安装软件。拓展打开远程系统，拓展CompactRIO，拓展设备和接口以及软件。在软件上右键单击选择添加/删除软件。

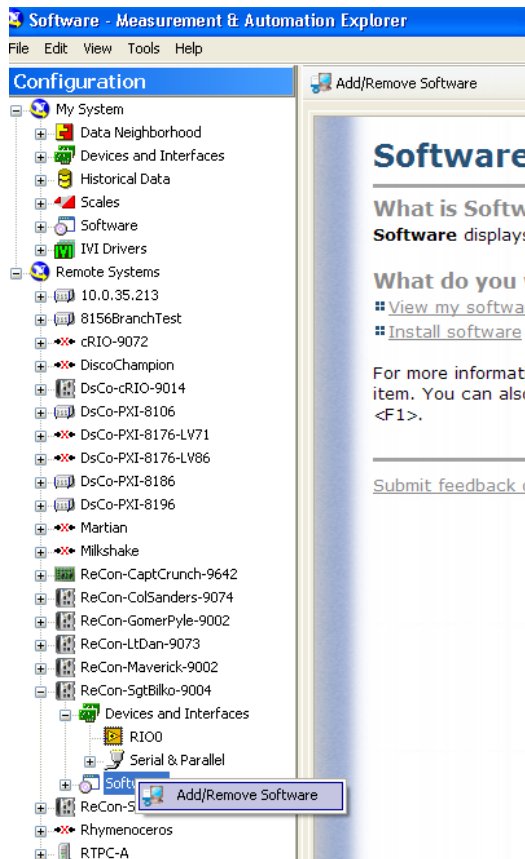


图 9.7 为CompactRIO系统安装软件

4. 软件会弹出一个安装向导。点击下一步，遵循提示安装软件。CompactRIO系统需要与主机电脑安装相同的软件，其中也包括附加组件，而且软件版本必须保持一致。软件安装完成后，CompactRIO系统自动重启。

现在可以将你的CompactRIO系统添加到LabVIEW项目中。

添加CompactRIO系统到LabVIEW项目中

遵照以下步骤，使用RIO扫描接口来建立一个LabVIEW项目。

1. 在LabVIEW开始窗口中，新建一个项目。

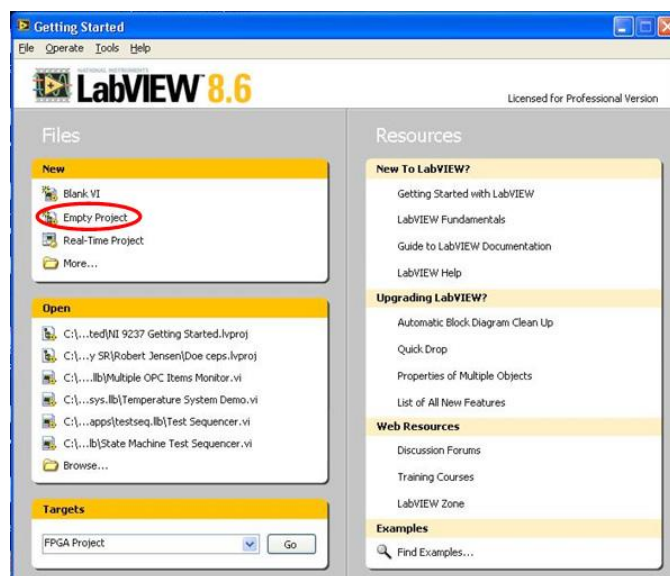


图 9.8 在LabVIEW开始窗口中，新建一个项目

2. 为项目命名并保存在一个文件夹中。在项目名称处右键单击选择**新建»目标和设备**。这时会自动弹出一个窗口，拓展打开 Real-Time CompactRIO，选择你的远程目标并添加到项目中。如果你还没有获得硬件，你可以选择“新建目标和设备”，手动选择硬件型号。

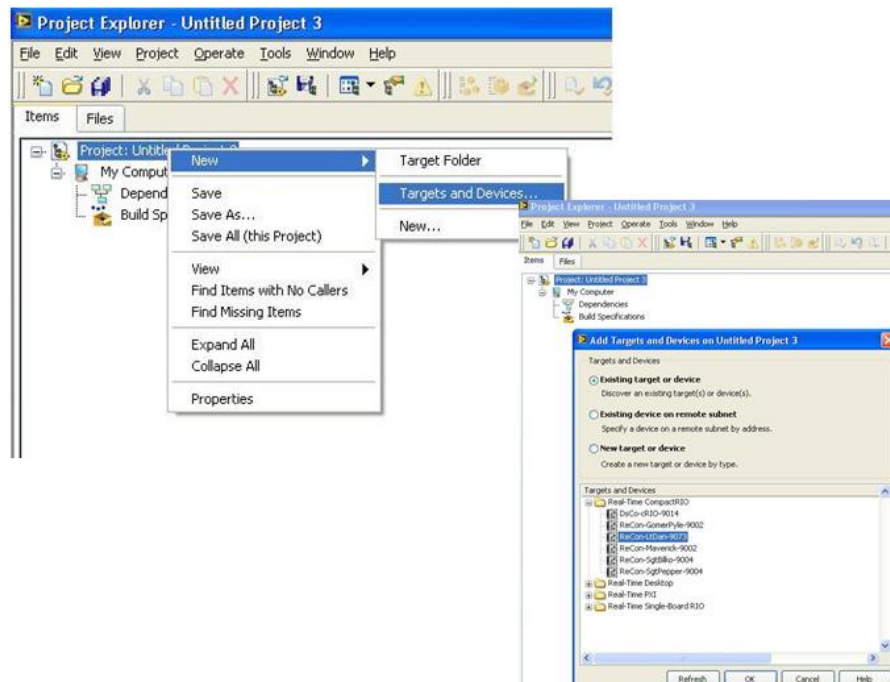


图 9.9 选择远程目标并添加到项目中

3. 如果你的CompactRIO控制器包含模块，会弹出一个对话框立即询问是否需要使用CompactRIO**扫描模式**或FPGA接口添加模块。如果你选择扫描模式，机箱和模块会自动加到你的项目。

你可以现在编制实时控制应用程序或者可以打开示例程序，开始体验。

修改一个既有的LabVIEW项目

如果你选择自定义一个示例程序来配合你的远程目标工作，可以在“LabVIEW项目例程浏览器”中的“实时目标例程”上右键单击选择“属性”。在“一般”栏目下，“IP Address/DNS Name”中修改IP地址，使其与你的CompactRIO保持一致。需要时常将例程另存为副本，选择**文件»另存为»复制项目文件和内容**，并将所有项目内容保存在另外一个单独的文件夹中。

附录B

LabVIEW调试工具

LabVIEW开发应用调试

当用户编写和开发编码时，LabVIEW提供许多工具帮助用户理解代码的执行。这些工具通过程序的执行过程来指导你，而且在调试过程中，可能会决定程序的应用。以下工具，位于程序框图的工具栏中，在应用调试时非常重要。

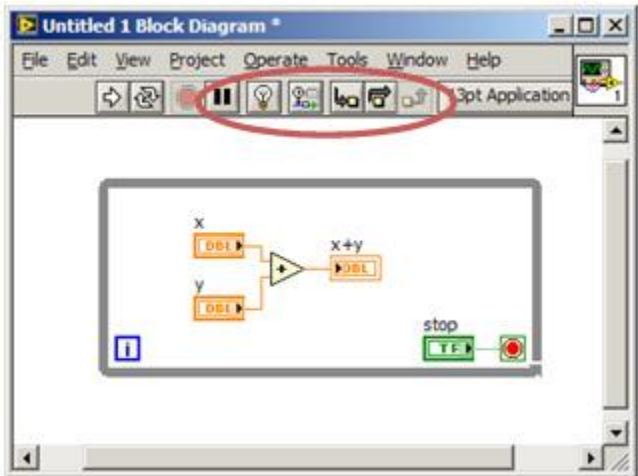


图 10.1 LabVIEW 具有多样的调试工具

Highlight Execution:

执行高亮显示程序框图并跟踪数据流，允许查看中间值。启用高亮显示执行过程可以单击工具栏上的 Highlight Execution。当点击白色的运行箭头，高亮执行显示数据在程序框图中运行状态，数据以气泡的形式沿着连线从一个节点运动到另一个节点。在单步操作中使用执行高亮查看数值如何在VI的节点之间传递。



Single-Stepping

单步执行的功能是在程序框图中显示VI程序运行的每一个动作。单步按钮仅仅在VI或子VI的单步模式下影响程序的执行。

在程序框图的工具栏中点击**Step Into**或**Step Over**键，开启单步模式。将光标移至**Step Into**，**Step Over**或**Step Out**键处，会提示如果点击此按钮下一步操作。在子VI中你可以单步执行或者正常执行。

当你在一个VI中选择单步执行，节点会闪烁，表示准备执行。如果你在执行高亮模式下单步执行整个VI，子VI的图标上会出现一个执行标志，表示他们目前正在运行。

Breakpoints

在VI、节点、连线位置可以放置断点工具来暂停程序执行。当你在连线位置上设置一个断点后，数据通过连线后，程序执行便暂时停止，暂停按钮变成红色。在程序框图内放置一个断点，以暂停程序所有节点的执行。框图边界闪烁红色表示断点已经安置完成。

VI在某个断点处暂停时，LabVIEW将把程序框图置于顶层显示，同时一个选取框将高亮显示含有断点的节点、连线或脚本。光标移动

到断点上时，“断点”工具光标的黑色区域变成白色。

程序执行到一个断点时，VI将暂停执行，同时暂停按钮显示为红色。VI的背景和边框开始闪烁，可进行下列操作。

- 用单步执行按钮单步执行程序。
- 查看连线上在VI运行前事先放置的探针的实时值。
- 如启用了保存连线值选项，则可在VI运行结束之后，查看连线上探针的实时值。
- 改变前面板控件的值
- 单击暂停按钮可继续运行到下一个断点处或直到VI运行结束。

可以使用断点管理器窗口禁用、启用、删除或布置已有断点。选择**View»Breakpoint Manager**打开断点管理器窗口，也可以在程序框图中右键单击目标选择**Breakpoint»Breakpoint Manager**，同样可以打开断点管理器窗口。

Probe Tool

使用探针工具可以查看VI中的即时值。如果程序框图较复杂，且包含一系列每步执行都可能返回错误值的操作，则可使用探针工具。将探针与高亮显示执行过程、单步执行和断点配合使用，可确认数据是否有误，并找到错误所在的位置。如有可用数据，高亮显示执行过程、单步调试或在断点处暂停时，探针都会立即更新和显示数据。当执行过程由于单步执行或断点而在某一节点处暂停，可用探针探测刚才执行的连线，查看流经该连线的数值。

Disabling Code

开发人员还可以禁用整节图形代码，这种方法类似于在语句编程语言中添加注释。禁用部分代码且认为程序除禁用部分外可以正常运行，是一种解决程序运行失败或编译错误的有效措施。在程序禁用结构中，Lab VIEW这种功能得到体现（如图10.2）。程序框图禁用结构包括一个或多个子程序框图，其中至少有一个框图包含禁用的代码，这部分代码在VI编译和运行时将不被执行。

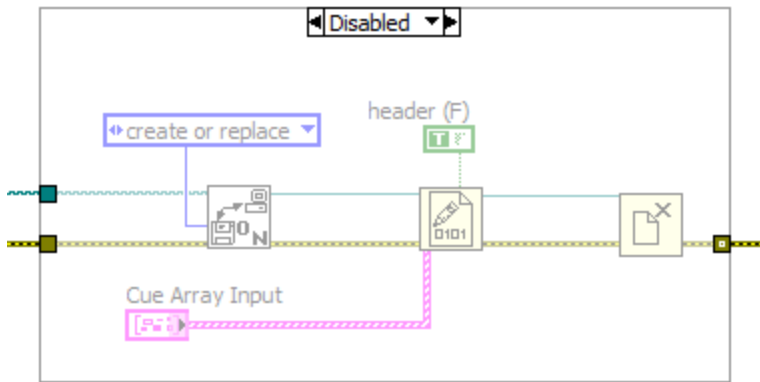


图 10.2 Lab VIEW程序框图禁用结构

调试已部署的应用程序

Debug Application Dialog

Lab VIEW中，同样可以调试内部的应用程序。调试内部应用程序，可以打开**Debug Application or Shared Library**对话框，调试运行于“实时”目标的独立“实时”应用程序。在Debug Application对话框中，同样可以查看程序框图，在高亮执行显示模式下运行程序，探针输入、输出，并且允许使用其他调试工具。在程序窗口点击**Operate»Debug Application or Shared Library**，可以打开Debug Application or Shared Library对话框。

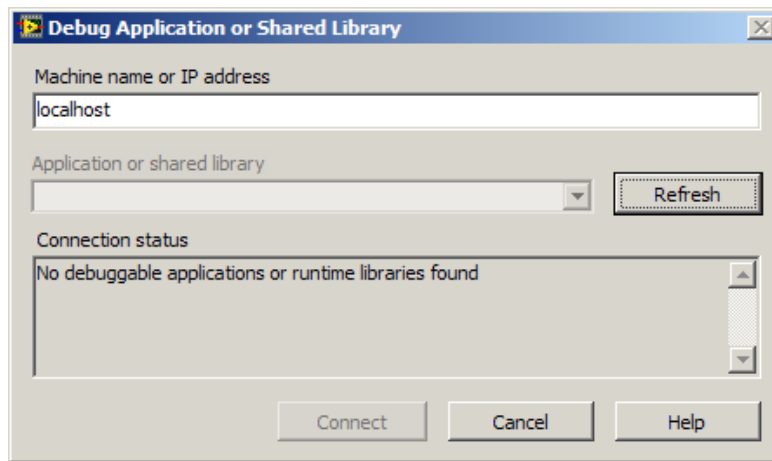


图 10.3 调试应用程序或共享库

使用应用程序调试功能前，需要更改Lab VIEW Application Builder中的选项。遵照以下步骤完成调试一个独立“实时”应用程序。

1. 在“实时”应用程序属性对话框中，转到“高级”页面，勾选“开启调试”，即开启“实时”应用程序的调试功能。
2. 右键单击，在快捷菜单中选择Build，建立一个独立的“实时”应用程序。
3. 重启“实时”目标，运行独立的“实时”应用程序。
4. 在程序窗口，选择**Operate» Debug Application or Shared Library**，弹出调试应用程序或共享库对话框。
5. 在机器名称或IP地址栏中为“实时”目标指定IP地址。点击“刷新”，查看“实时”目标应用程序列表。
6. 选择需要调试的独立“实时”程序。
7. 单击“连接”按钮，开启调试VI前面板。在程序框图中，运用探针、断点、以及其他调试工具，对应用程序进行调试。
8. 调试结束后，关闭调试VI，此操作同时关闭“实时”目标中的独立“实时”应用程序，重启“实时”目标后，独立“实时”应用程序也将重启。如果需要断开连接，且不希望关闭调试VI，在调试VI前面板右键单击，快捷菜单中选择**Remote Debugging» Quit Debug Session**。

NI Distributed System Manager

NI分布式系统管理器提供了一个网络中央监测平台，用于查看、清除错误，并且管理发布的数据。NI分布式系统管理器是一个独立工具，不需要与Lab VIEW安装于同一台电脑。可以从开始菜单进入NI分布式系统管理器，也可以在“实时”目标项目浏览窗口中选择**Tools» Distributed System Manager**。

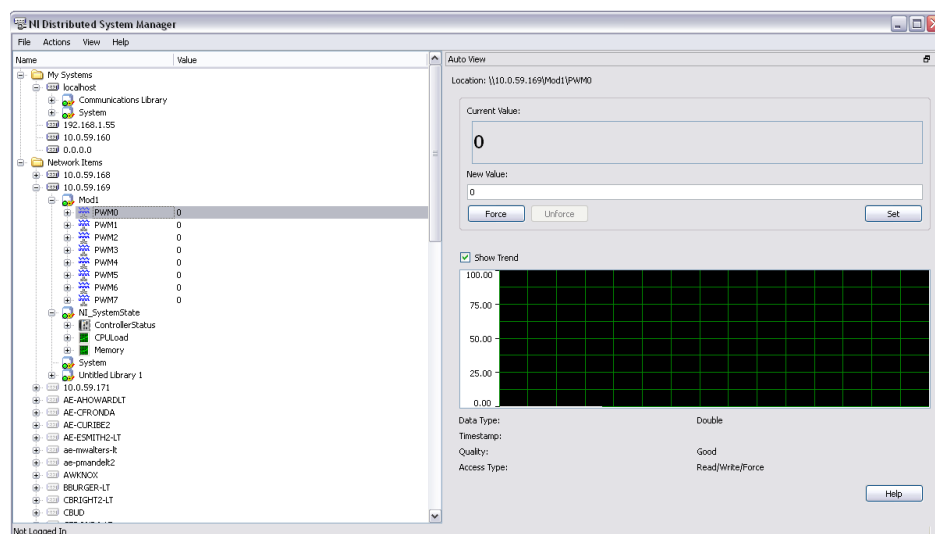


图10.4 利用NI分布式系统管理器查看I/O及控件状态

在扫描模式下，系统管理器为Compact RIO模块提供了测试面板，当你的系统可用时，可以查看“实时”数值以及历史数值，帮助检查连接状态和信号完备性。与此同时，系统管理器可以显示Compact RIO控制器内存和处理器的使用率。同样可以利用NI分布式系统管理器来监视、管理扫描引擎的故障和模式。

User-Controlled LED

大部分的“实时”控制器都有一个或多个指示灯，由用户程序来控制。程序设计者可以利用指示灯显示程序运行状态。通常，利用指示灯的闪烁频率来表征程序目前运行与何种状态。Lab VIEW专为“实时”目标开发了一个“实时”指示灯VI，帮助用户更好的交流。

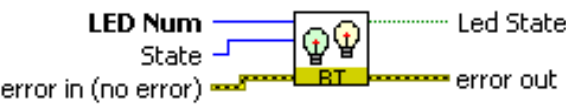


图 10.5. 指示灯可以简单表示程序运行状态

在“实时”指示灯VI程序框图中，按以下表格设置状态输入值，来改变指示灯状态。

0	开启指示灯
1	设置指示灯为默认颜色1
2	设置指示灯为默认颜色2
3	在关闭和默认颜色1之间转换

表10.1. 指示灯状态输入

不仅如此，一些“实时”控制器为FPGA应用程序提供指示灯接口。使用默认以FPGA LED命名的Lab VIEW FPGA I/O节点，编写Lab VIEW FPGA应用程序，根据布尔控值，开启指示灯，并显示其状态的改变。

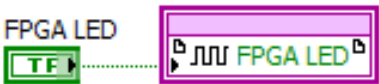


图10.6. 也可由FPGA进入指示灯功能

Compact RIO Console Out Switch

Compact RIO控制器提供了一个控制台输出开关，当控制器通过串行端口连接至电脑时，由COM1端口反馈到有用的诊断信息。此项功能在下列情况的系统故障诊断中极为有用。

- 在控制器中，将写格式化输入样式的信息显示至文本控制台。
- 显示控制器的当前固件版本号和IP地址。
- 诊断一个无响应或指示灯显示错误的控制器。
- 对于在MAX中无法显示的控制器进行故障诊断。
- 协助NI Applications Engineer进行控制器故障诊断。

查看Compact RIO控制器控制台输出值，请遵照下列步骤：

1. 关闭控制器。
2. 通过非调制解调电缆将控制器连接至PC（NI控制器属于DTE设备）。
3. 在PC上，应用以下串行接口设置：
 - a. 每秒位数：9600
 - b. 数据位数：8
 - c. 校验：无

- d. 停止位：1
- e. 流控制：无

4. 调整控制器上的Console OUT DIP开关处于ON的位置。
5. 打开控制器，查看PC终端窗口输出值。

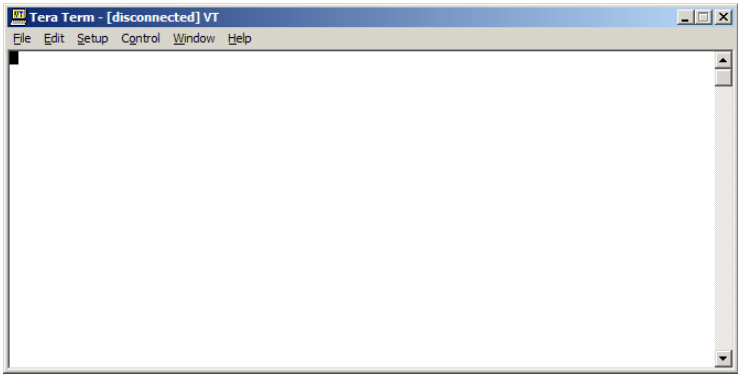


图10.7. 控制台输出为控制器提供了一个简单的串联接口

6. 调整Console OUT 开关回到OFF位置，完成故障诊断。

Lab VIEW “实时” 为控制器提供了一个“实时” 调试字符VI，可以将对话文本信息由控制器发送至主机PC终端窗口，从而反馈得到控制台调试信息。

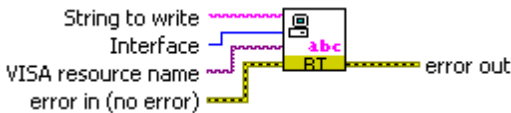


图10.8. 通过自定义应用程序和控制台输出发送附加数据至PC

Error Logging

Compact RIO程序通常没有用户界面，因此无法确切得知错误何时发生。因此，在控制器日志文件中保存这些错误就显得极为重要，以便需要时调出查看，其同样可以保存LabVIEW错误信息和难以预料的数据信息。通常建立一个独立的更低优先级的循环结构，记录错误，或者依据此错误来关闭程序。如下为一个简单的错误日志结构。

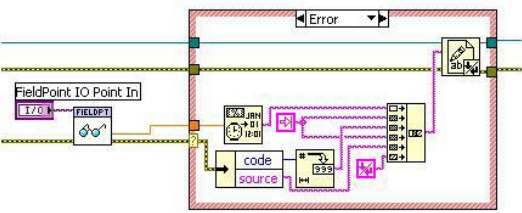


图 10.9 简单错误日志结构

除此之外，如果“实时”控制器崩溃，其自动生成一个错误日志，在MAX的远程系统中，右键单击控制器，选择查看错误日志，可以查看错误日志。错误日志文件位于控制器中的下列目录：/ni-rt/system/errlog.txt.

Lab VIEW 分析工具

建议在开发应用程序时，分析你的VI，以获取程序执行时间以及检查不必要的操作，如内存分配和线程转换切换。

分析VI和代码段

Real-Time Benchmarking VI

对操作进行标准检查包括：确定程序开始执行时间、执行程序 and 程序完成执行时间。“计时”在Lab VIEW中扮演很重要的角色，“实时”功能搭载了一个标准检查VI，帮助确定准确的计时操作。

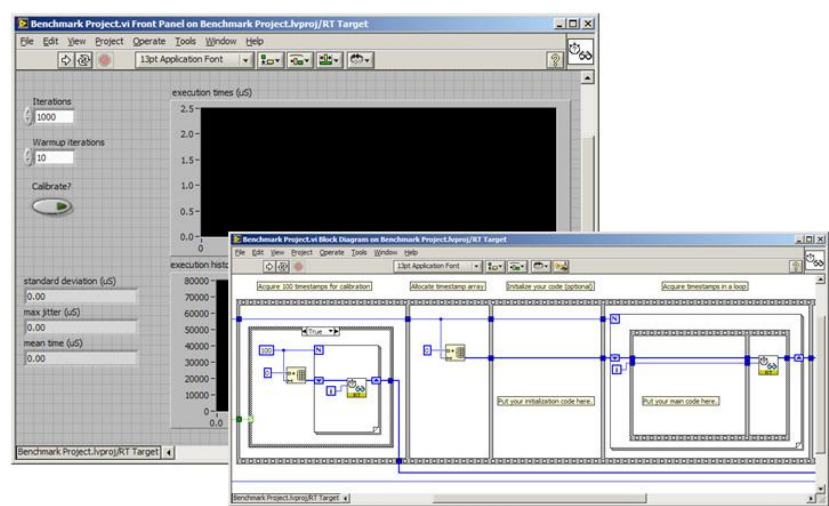


图10.10. The Real-Time Benchmark VI 能够帮助了解子VI的执行时间

Benchmark Project VI中调用了RT Get Timestamp VI和RT Timestamp Analysis VI，检查目标VI的运行情况。利用标准检查信息，优化“实时”目标程序设计。可以利用NI范例查找器，打开“实时”标准检查VI。

Profile Performance and Memory Window

Lab VIEW为用户提供了一个性能和内存信息窗口，以监测VI内存占用和运行时间。它可以显示内存中所有VI和子VI的运行状态，通过发现潜在的瓶颈来优化VI。例如，如果发现某一子VI占用运行时间过长，便可以对其进行改进。

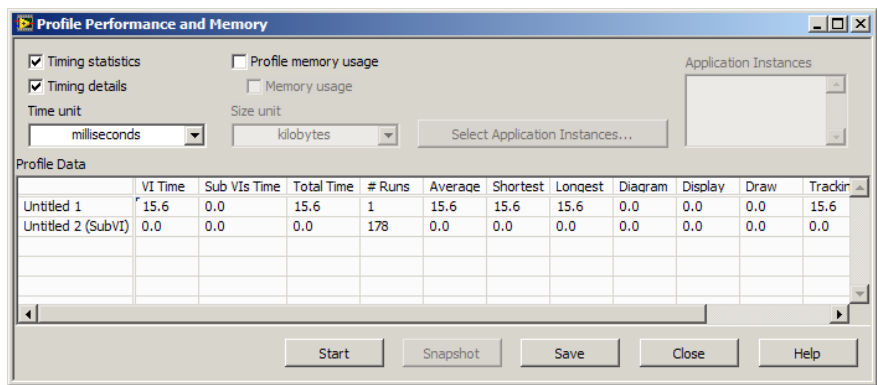


图 10.11. 启动性能和内存信息窗口提供了应用程序和子VI的运行时间和占用内存的具体情况

启动性能和内存信息窗口，可以选择**Tools» Profile» Performance and Memory**。

监测Real Time中的目标资源

NI Distributed System Manager and Real-Time System Manager

控制器运行已部署的程序代码时，建议查看其内存和CPU的使用情况。一些情况下，由于real-time目标的内存和CPU资源不足，造成计时程序无效。监测目标资源可以知道是否发生了内存溢出（通常由于在循环结构中分配内存造成的），以及各循环结构占用CPU的情况。利用NI分布式系统管理器可以查看控制器的CPU和内存的使用情况。如图10.12.所示。

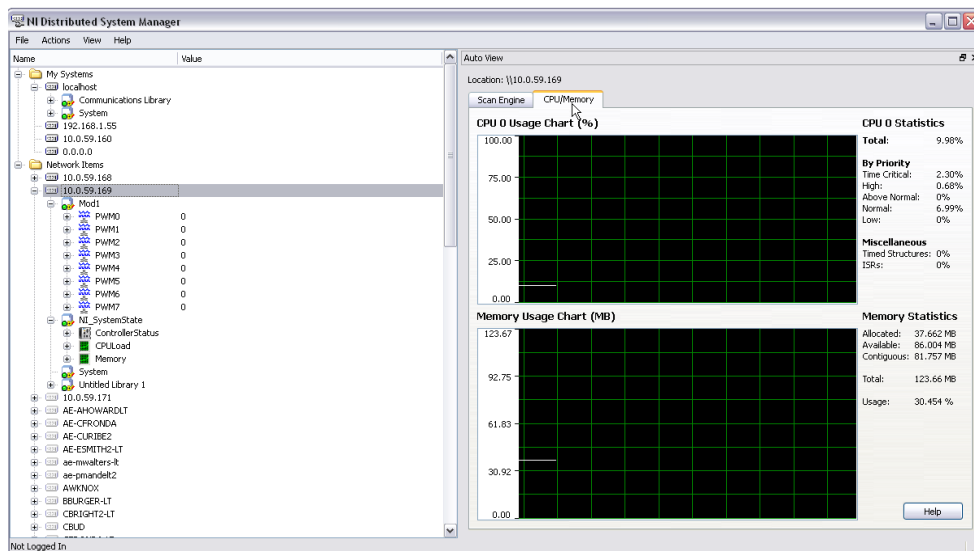


Figure 10.12. NI分布式系统管理器显示了网络上控制器的CPU和内存的使用情况

也可以使用NI Real-Time系统管理器查看CPU和内存使用情况。

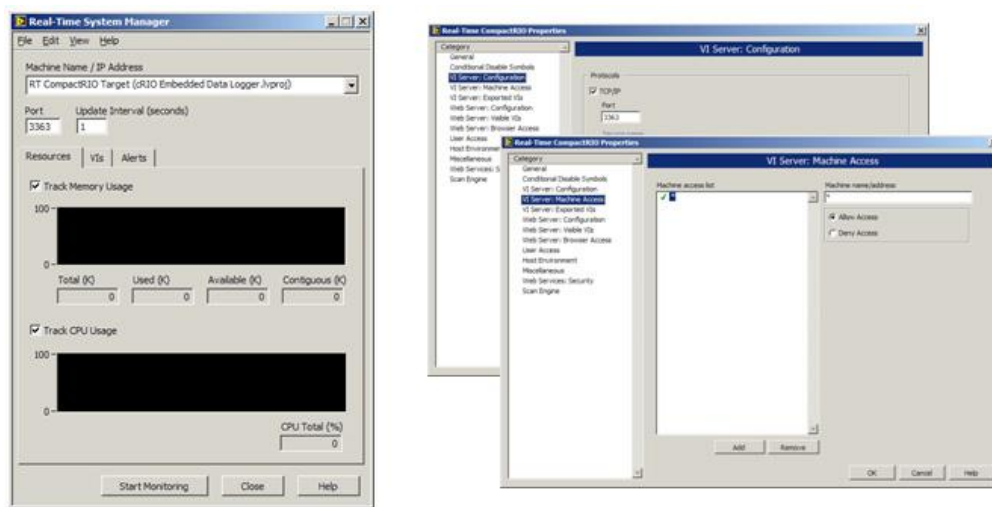


Figure 10.13. NI Real-Time 系统管理器显示了CPU和内存得使用情况

在Tools» Real-Time Module菜单中启动NI Real-Time系统管理器。需要事先配置好VI服务器，才可以设置NI Real-Time系统管理器。配置VI服务器，在real-time目标上右键单击选择属性，打开Real-Time属性页面。点击弹出VI服务器配置页面，勾选TCP/IP，同时勾选可用服务器资源中的选项。在VI服务器机器访问一栏中，确保其会显示你的PC主机IP地址或“*”符号。

Real-Time Execution Trace Toolkit

Execution Trace Tool 是最有效的工具之一。在Lab VIEW实时应用程序中，执行轨迹工具作为实时事件，用以获取和显示VI的计时和事件数据，完成连线工作。可以利用执行轨迹工具VI获取数据的计时和执行情况，并为运行于“实时”目标上的应用程序添加连线。

这些工具，能够最大程度的保持嵌入代码不被修改，同时以图形的方式高亮显示代码执行情况，包括线程互换、互斥和内存分配等。利用这些信息，可以优化程序、提升循环速度，使程序运行更加明确。

在“实时”目标程序中，添加执行轨迹工具VI。当追踪完成时，这些VI将信息反馈到Execution Trace Tool的主机用户界面中，或者将信息保存用以日后查看。下面的例程可以看出，在被关注的代码前面加入Trace Tool Start Trace VI，开始追踪；在后面加入Trace Tool Stop Trace and Send VI，将代码执行后的信息反馈到主机。

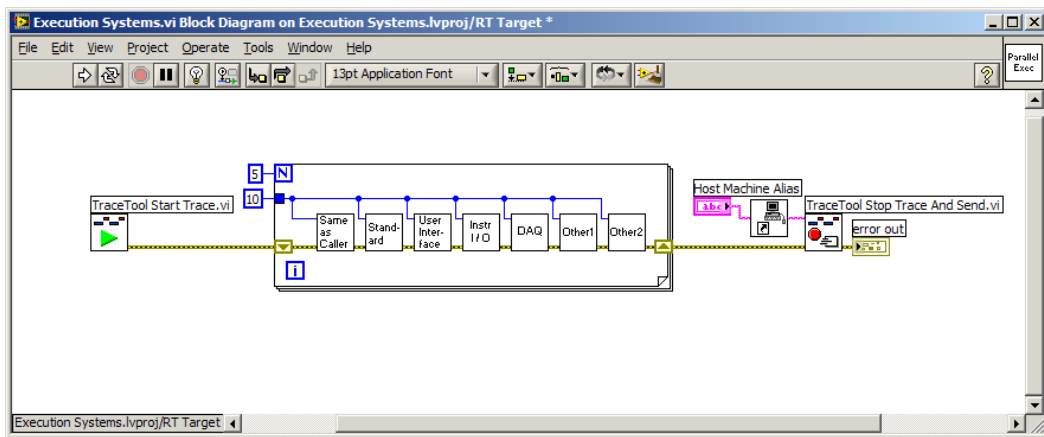


图 10.14. 通过修改程序使其进行轨迹记录并将记录发送到PC上，这样就可以看到程序执行的底层细节

Execution Trace Tool的主机用户界面接收到信息后，可以看到两个同步追踪显示单元。线程显示窗口显示当前执行的的线程。VI显示窗口显示在内存中的VI并显示其执行的确切时间。

线程显示窗口显示所有在系统中运行的线程。其中也包括了Lab VIEW以外的线程，例如“实时”计时和交互进程。Lab VIEW线程，用彩色标记优先在VI窗口中显示，并根据Lab VIEW所处的执行系统为其添加标签。线程显示窗口还可以显示操作系统、定时循环、和用户标记等内容。

正如之前所提到的，VI显示窗口显示在内存中的VI并显示其执行的确切时间。然而，VI显示窗口中两个VI可以重叠，但是其确切的执行时间是不同的，具有高优先级的VI会优先于另一个VI先执行，此时低优先级VI无法占用处理器。这种切换非常迅速的执行完毕，以至于无法判断低优先级VI是否处于未执行状态。在线程显示窗口中，可以查看哪个线程正在执行，表示相应的VI正在执行。

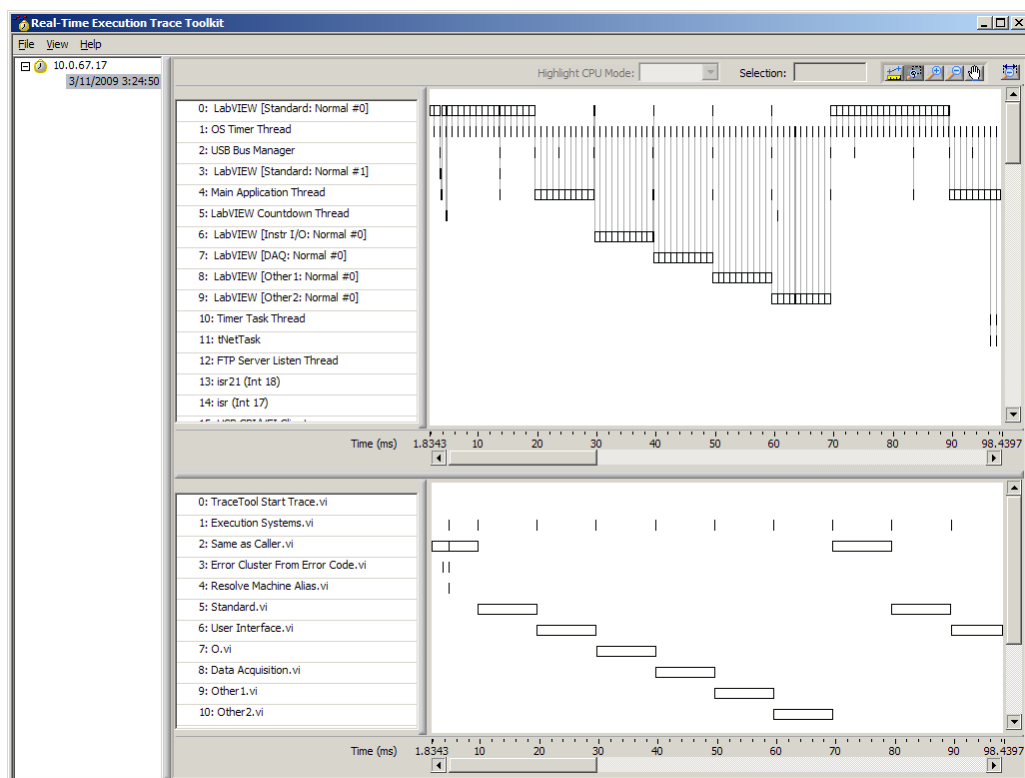


图10.15. Execution Trace Tool提供了应用程序执行的底层细节，显示了正在执行的线程以及优先权和内存的分配情况

通常，用户需要查看除VI开始和停止以外的事件，例如，进入内存管理器、睡眠、定时循环迭代等。可以使用Real-Time Execution Trace Toolkit根据事件类型来设置和标记事件，在Execution Trace Tool点击**View» Configure Flags**打开该功能。参阅Lab VIEW帮助以取得更多详细的资讯。

调试Lab VIEW FPGA

开发Lab VIEW FPGA与开发Lab VIEW Real-Time及其相似。Lab VIEW FPGA中相同的图形程序架构适用于所有平台，并且其大部分函数为Real-Time函数的附属函数。然而，为优化你的Lab VIEW FPGA程序，可以采用不同方法来平衡你的硬件概念，例如寄存器、流水线、时钟周期、异步模块叫唤。

开发FPGA与开发Lab VIEW Real-Time有何不同？

FPGA的硬件开发技术与软件开发不同，主要体现在以下三个方面。

编译时间

FPGA可执行代码编译复杂，需要满足芯片中逻辑结构高度优化。程序编译时间取决于程序的复杂程度。

无典型Lab VIEW硬件调试特性

硬件程序运行时，无法使用传统的软件调试工具，如探针、单步执行、执行高亮、设置断点。

FPGA执行迅速稳定

FPGA作为应用程序，需要其高速执行、稳定执行。其次，程序员通常关心FPGA每个时钟周期的运行情况，以便完善程序，也可以理解为了解平行任务的同步情况。

程序编译的编译时间为5分钟至几个小时不等，因此选择“编码与修复”编程方法不切实际。要想成为熟练的FPGA程序员，提升你的开发效率就显得极为重要。下面一节介绍了如何利用性能模拟技术以及“后编译”调试技术创建一个稳定、快速的FPGA系统。

Lab VIEW FPGA “后编译”调试技术

利用输入/显示控件进行调试

程序运行于FPGA中时，输入控件和显示控件为代码调试提供了很好的架构体系。因为无法使用Lab VIEW探针工具，所以利用显示控件可以得到及时的目标值。除此之外，由于改变常值需要重新编译，所以通常利用输入控件代替常值。如果利用输入控件得到你认为正确的常值，可以将其替换为常值。然而，输入、显示控件与主机VI需要接口数据交换，其占用FPGA中相当一部分资源，因此资源优化的第一步就是要去掉不必要的输入、显示控件。调试阶段可以增加输入、显示控件，但是程序设计完成时需要去掉多余的调试控件。

锁存错误状态

许多情况下，错误状态仅仅发生在很短的时间内。最坏的情况是错误仅仅发生在一个时钟周期内。调试过程中，这些错误状态迅速产生以至于无法在交互界面中将其扑捉到。捕捉错误的最佳方式是当错误发生时将其立即锁存，之后如果需要允许其复位。得到正确的运行结果后，打开调试锁存器，在单独的程序中处理错误。下面是一个调试FIFO周期超时的经典例程。

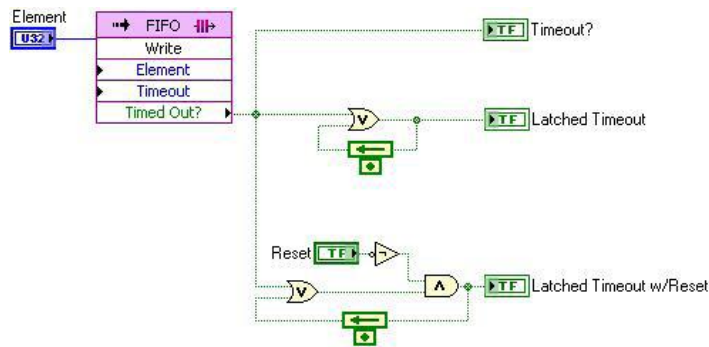


图10.16这个VI显示了如何锁存超时状态

DMA方式记录测试点数据

有时，需要在FPGA中添加探针，但是输入控件产生的瞬时值不足以满足调试需要。这种情况下，可以利用DMA通道实际捕捉测试点的整个波形数据，满足调试需要。

发送测试点数据至I/O

无论是模拟测试点还是数字测试点，总是可以将Lab VIEW连线发送至I/O节点。一旦FPGA产生信号，可以立即利用任何一个NI测试硬件调试信号，例如显示器、数字板卡、DAQ卡，或者还可以利用另外的FPGA调试程序进行调试。

利用条件结构测试多个候选执行

一些情况下，需要测试FPGA中任务执行的不同方式。可以依次编译测试哪种执行效果更好。然而，为避免额外的编译工作，可以将测试程序放入一个条件结构中，选择不同条件，通过对比不同条件得到及时反馈情况，避免了额外的编译时间。

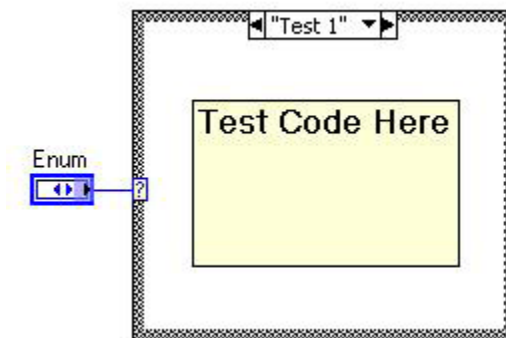


图10.17将想要在FPGA上测试的不同代码放置在不同的条件结构中

位于开发计算机上的FPGA性能模拟

创建FPGA逻辑时，多数人仅仅使用了Lab VIEW代码，因此常常在上位机中执行目标VI。这意味着，如果你需要测试一些逻辑，完全可以调用所有的调试特性而不必等待其编译结果。除此之外，可以创建一个测试VI，将“输入”通过用户自定义I/O与外部相连，得到输出值，用于分析验证。最后，同时运行上位机程序以及包括模拟寄存器和DMA FIFO存储缓存的FPGA代码。

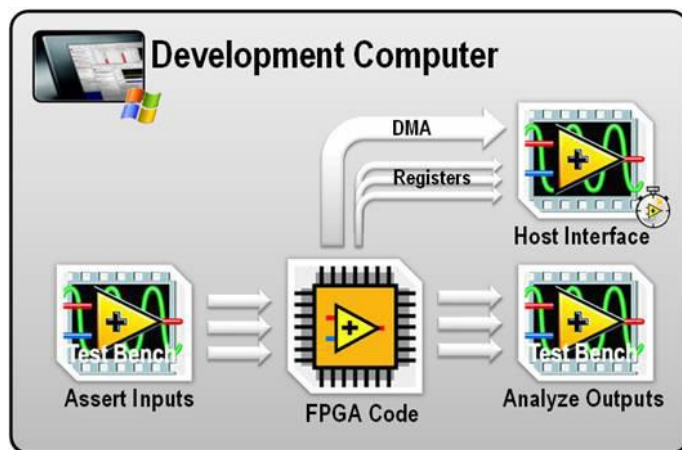


图10.18使用Testbench VI的模拟系统概念框图提供了拥有主界面的自定义输入和输出

设置即将运行于开发计算机的FPGA，需要在项目中右键单击FPGA，选择“属性”。在“调试”一项中，有三个选项：

1. **Execute VI on FPGA target** –如选择此项，当点击FPGA VI的运行箭头时，立即开启编译进程。
2. **Execute VI on the development computer with simulated I/O** –该选项将FPGA VI设定为在PC上运行。在下拉菜单中，可以选择使用任意数据（Lab VIEW 8.5或更早版本的主要特性），也可以选择自定义I/O，这样就可以编写一个测试VI来指定输入，同时捕捉输出。
3. **Execute VI on the development computer with real I/O** –该特性只适用于R系列插入式设备。该选项将使VI在PC上运行，下载固定特性至R系列设备来扫描程序执行的I/O节点。该选项适用于早期测试和原型设计，需要注意的是，I/O扫描时钟是由软件控制，并不代表你所希望使用的VI时钟。

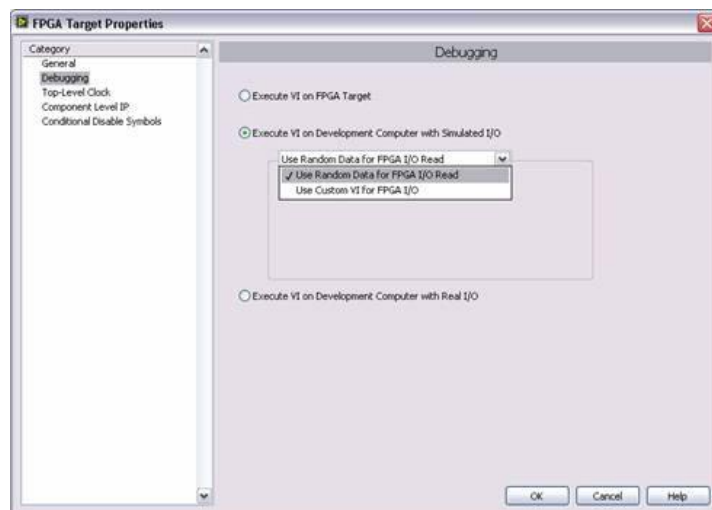


图10.19FPGA属性窗口提供了执行FPGA代码及I/O类型时的三个选项

附录C

大型程序开发 最佳实践

大型程序开发指南

适当的软件工程实践对于成功开发**Compact RIO**控制系统极为重要。任何工程都应以明确定义的需求作为开发起点，需求决定程序架构，针对架构开发应用程序，对违背需求的程序代码进行修改完善。

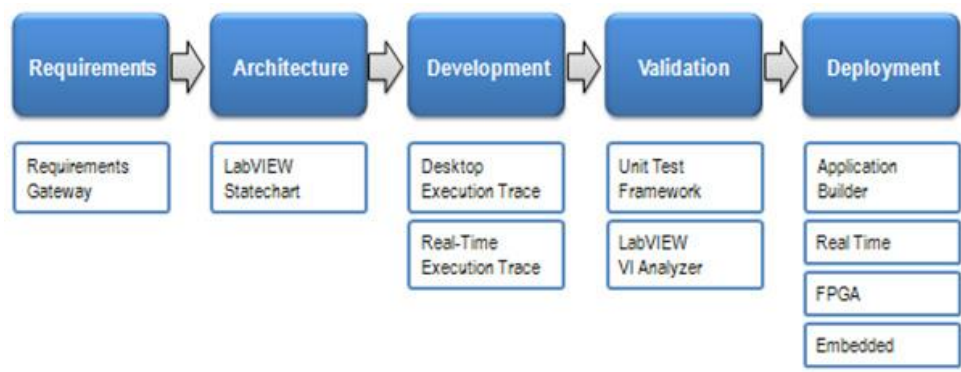


图11.1适当的软件工程实践对于成功地完成项目极为重要，NI公司提供了一些工具来帮助执行每一步

在**Lab VIEW**中建立程序相对比较容易，因此多数人仅仅注重程序开发步骤而忽略了程序规划步骤。当然对于简单的应用程序，如快速试验程序或监测程序，也可以满足要求，但是对于大型程序开发项目来说，程序规划就显示极为重要。这一节主要介绍了利用**Lab VIEW**成功开发软件的一些基本技巧。参阅**Lab VIEW开发指导手册**以取得更多详细的资讯。

应用结构化软件工程思想

程序设计人员开发的基于任务的关键应用程序（嵌入式控制程序、工业监测程序、高性能测试系统）必须保证无错稳定运行。针对这几种应用程序，工程师需要遵循有条理的、甚至外部程序流程来保证代码质量和可重复性。一般来说，针对下列情况，工程师需要遵循结构化程序设计流程：

- 创建大型应用程序
- 作为团队合作开发的程序，需要结构化的程序设计思想，避免一个项目中的不同领域发生冲突，确保团队成员的工作效率。
- 作为需要政府机构（**FDA**，**FAA**等）或用户（汽车制造商）认证的应用程序。

很多工程师可能认为这些环境需要基于文本的传统编程语言代码。事实上，一个结构化的代码开发过程（无论使用何种任何语言）是独立于编程语言或使用的工具。这一章节概述了常用的程序开发方法、内置工具、以及外部整合结果，其中外部整合结果向工程师展示了如何创建大型或需要结构化程序设计方法的关键任务系统。

其中外部整合结果向工程师展示了如何创建大型或需要结构化程序设计方法的关键任务系统。

获取需求和管理

掌握系统的需求是开发任何大型程序开始之前的一项关键步骤。系统开发人员使用许多不同的方法获取并记录系统需求，可以是简单

的Microsoft Word或Excel，也可以是需求管理工具例如Telelogic DOORS和IBM Rational RequisitePro。管理需求的关键是确切的记录需求的内容并且使其与执行代码捆绑。

为方便用户获得系统需求，NI公司开发了NI Requirements Gateway功能软件，工程师可以利用此软件将需求与LabVIEW VI进行关联，追踪执行结果。利用Gateway可以进行如下操作：

- 利用文本文件、Microsoft Word、Excel、项目、Telelogic DOORS、IBM RequisitePro以及其他工具记录系统需求，也可以利用toolkit附带的需求管理器记录系统需求。
- 将各需求与指定的LabVIEW VI或一段代码进行关联，追踪各需求的执行或测试情况。
- 追踪需求的变化情况。利用NI Requirements Gateway，工程师可以快速查看需求的变化，提示与需求相关联度的VI，确保其始终与需求相关联。

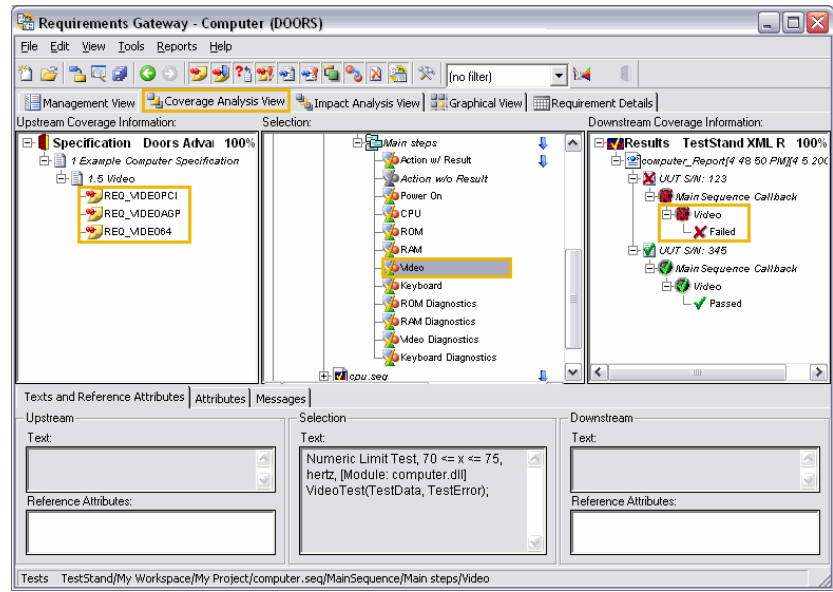


图11.2. NI Requirements Gateway

架构规划与设计

这一节提出了一种可扩展程序架构，适合多控件应用程序。然而，顺利地建立程序架构需要程序设计者针对程序需求规划好软件框架，规划工作需要完成定义系统组件及其之间的交互作用。最后，开发人员需要将系统需求转化为可执行的程序代码。

利用Lab VIEW开发大型应用程序与其他程序设计软件并无差别，需要将程序划分为容易处理的逻辑组成。Lab VIEW图形化编程语言将使程序设计变得简单、直观。除图形化编程外，Lab VIEW同样提供了解决问题的具体方法，包括：

- 状态机
- 模拟框图

开发

管理磁盘文件

磁盘文件管理是最后需要考虑的步骤。对大型程序来说，如果规划不得当，会导致开发期间移动和重命名文件操作占用额外的时间。Lab VIEW VI是根据其名称和路径进行关联，因此如果你移动或重命名子VI，关联就会遭到破坏，需要手动重新建立关联。适当管理磁盘上的文件会降低日后移动大量文件的风险，帮助开发人员很容易的找到文件，并决定在哪里保存新文件。

许多软件开发已经规定好文件存储的位置，但是除经典的方法和结构外，以下方法被证实非常适合大型程序的开发工作。

- 将所有项目文件存储到一个单一目录

- 在该目录下创建包含文件逻辑群体的文件夹
- 根据预定的标准对文件进行分组
- 将程序划分为易处理的逻辑单元
- 使用具有逻辑性和描述性的命名规则
- 将顶层VI与其他源代码区分开来

文件夹通常用来对文件进行分组、分类，因此可以针对不同的调用对子VI进行分组、分类。文件分组的原则是根据程序中文件的功能、类型以及分级层来进行的。实际上，磁盘管理真正体现了程序中文件和代码之间的关系。

使用“Lab VIEW Project”管理文件

Lab VIEW Project为开发人员提供了用于管理文件的工具。随着程序不断壮大，开发人员需要对程序关联文件进行管理，如VI、文件材料、第三方函数库、数据文件以及硬件配置文件。工程师可以利用Lab VIEW Project Explorer管理这些文件。

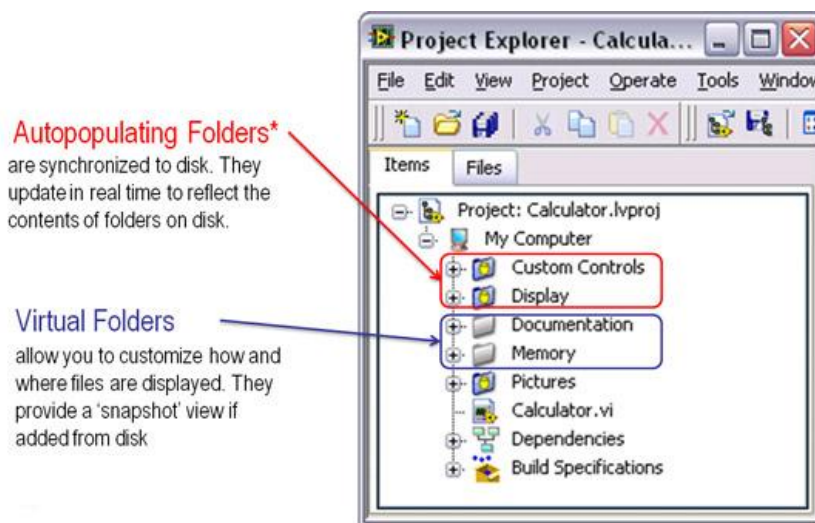


图11.4 Lab VIEW项目上的源代码选项

开发者可以利用Lab VIEW Project管理所有程序关联文件。默认的项目文件夹为虚拟文件夹，但是开发者可以将其与系统物理目录进行同步。一旦开发者在Lab VIEW Project中添加了一个目录，可以将其转变为“自动填充” (反之亦然)，以最大限度地提高文件管理和组织灵活性。自动填充文件夹会将磁盘文件管理与Project中的逻辑分组进行关联。如果可能，最好使用自动填充文件夹来保护Lab VIEW项目浏览器中的磁盘框架。

团队开发

1. 通常多个开发人员在同一项目中使用Lab VIEW进行合作开发。在这种情况下，明确代码访问接口和识别编码标准就显得极为重要，这样可以确保不同组件易于理解和相互配合。为此，开发人员可以将代码模块化并提供标准接口。
2. 在独立服务器中对项目文件（包括VI）进行备份，并制定一个源代码控制政策
3. 在小组或公司中执行编码标准

Lab VIEW代码模块化

大型Lab VIEW 程序应采用模块化开发方法，整个系统被划分为部分逻辑组件。多人开发项目需要这种模块化编程方法以明确各自责任。Lab VIEW作为模块化编程语言，其每个VI（或代码单元）可以作为一个独立的应用程序或作为另一个VI的子VI。在1131编程方式中，子VI所对应的子程序呼叫基于文本的编程语言或函数/函数块。大型程序中采用模块化子VI的方式简化了高层VI框图并且有助于变更管理和系统调试。

利用源代码控制软件共享代码

软件开发项目开始阶段，工程师必须开启变更处理进程并分享工作任务。这个过程对多人协作开发是非常关键的。源代码控制工具是解决这一问题的最好方法，其能够共享代码并控制访问，从而避免意外数据丢失。源代码控制软件跟踪代码模块的变更情况以及多用

户、多对象共享设备文件的使用情况。除了维护像VI这类源代码以外，源代码控制软件还可以管理软件项目的其他方面，如特性说明和其他文件。

Lab VIEW集成了许多工业级源代码控制软件，如**Microsoft Visual SourceSafe**、Perforce、**IBM Rational ClearCase**、PVCS Version Manager、MKS Source Integrity、以及免费开源软件**CVS**。通过集成源代码控制软件，开发人员可以在Lab VIEW内部环境中迁入或迁出源代码控制文件，并且获取文件的更改历史。源代码控制软件设置完成后，需要在Lab VIEW中迁入或迁出VI时，开发人员可以在Lab VIEW 项目浏览器中的文件上右键单击，在弹出的菜单中选择适当的操作。

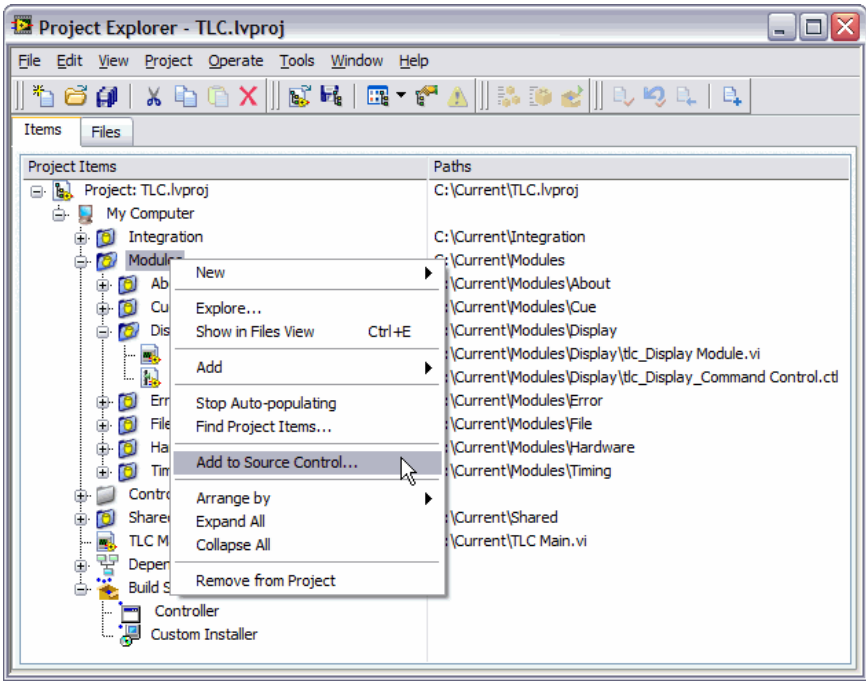


图11.4 Lab VIEW项目上的源代码选项

调试

调试功能对于开发环境至关重要，LabVIEW中包含多种调试工具，如：

- .. 执行过程高亮，显示表明了程序框图上的数据通过沿着连线移动的圆点从一个节点移动到另一个节点的过程。
- .. 单步执行VI，显示VI运行时程序框图上VI的每次执行过程。
- .. 断点，可以放置在程序框图VI、节点或连线上，用于暂停某个位置的执行过程
- .. 探针，显示VI运行时连线上的实时值

本文还有独立的一部分内容详细介绍LabVIEW实时和FPGA应用调试的工具与技巧。

验证与审查代码

欢迎开发人员使用LabVIEW来创建大型应用，并提交代码参与互评。 当开发人员开始代码审查时，他们带领审查者回顾代码主要路径，并回答相应问题。 讨论的议题包括 代码架构如何使得添加新功能或实施变化变得轻松，如何报告和处理错误以及代码是否足够模块化。

当准备代码审查时，开发人员需要采用能够自动进行代码检查并辨别改进的工具。 一个范例就是**LabVIEW VI分析仪**，它能分析LabVIEW代码并显示测试错误结果。 VI分析仪包含了60多个测试，可以解决众多代码性能和风格问题。 借助VI分析仪开发者可以提高VI性能、可用性和维护性。 VI分析仪还能够生成报告，因此开发人员可以追踪代码的改进。

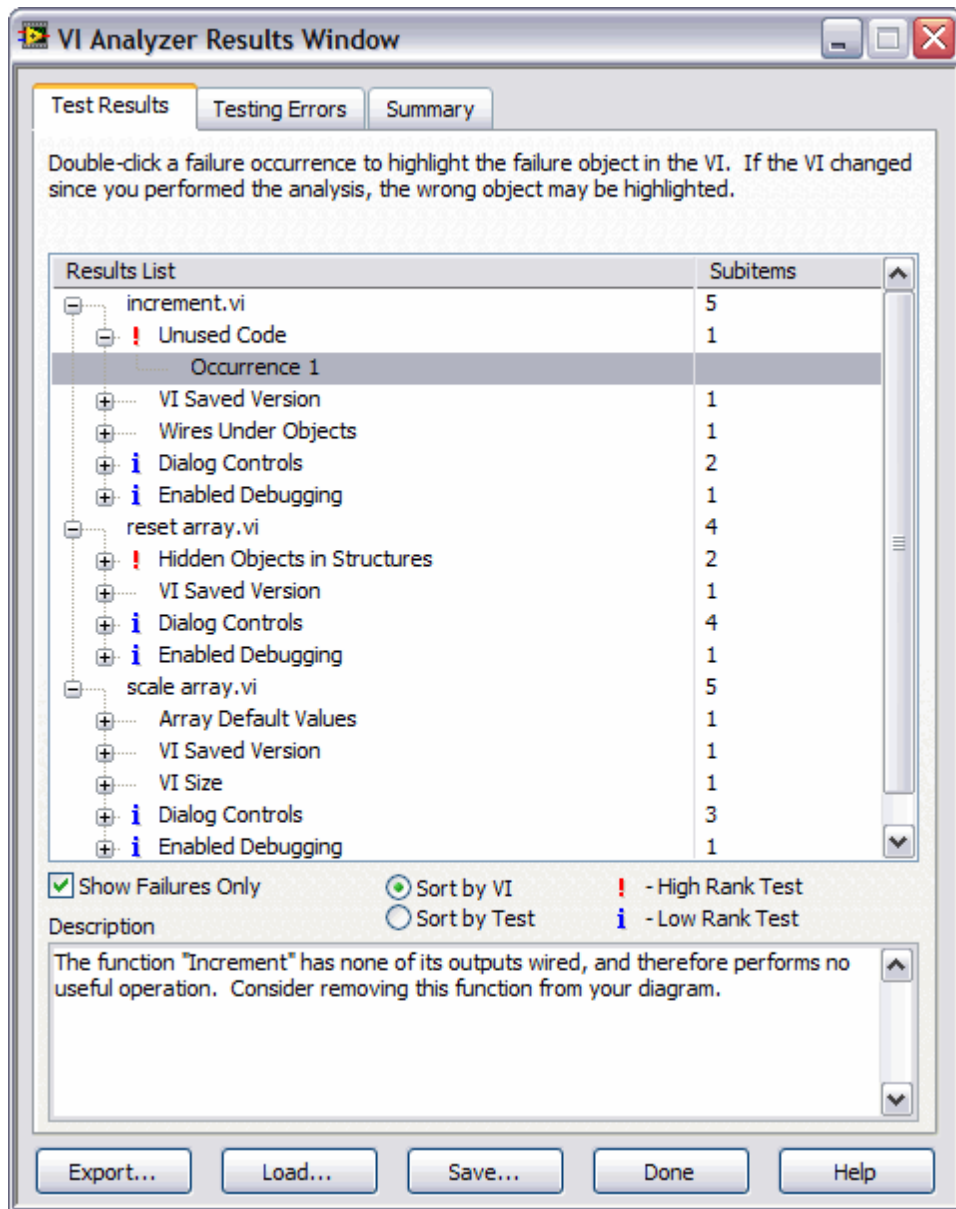


图11.5. LabVIEW VI分析仪

部署

控制应用的最后一步就是在嵌入式系统中部署应用并设置自动运行。要使LabVIEW VI启动时自动运行需要为实时目标创建exe文件。在LabVIEW项目中创建一个程序生成规范即可完成，其中包括可执行文件以及想要导出的支持文件上的所有信息。程序生成规范使得重新创建或自定制程序变得容易。LabVIEW也有系统复制工具，可以用来将控制器上的嵌入式代码复制另一个系统上。

