第 3 章

CHAPTER 3

虚拟仪器编程进阶

3.1 信息显示技术

本节主要介绍 LabVIEW 提供的波形图、波形图表、XY 图等信号显示控件及其在虚拟仪器程序设计中的应用。

3.1.1 常用的显示技术

丰富、强大的图形显示功能是 LabVIEW 开发平台的巨大优势之一。利用图形或者图表等形式对测量、测试数据及其分析结果进行可视化表达，可以极大地方便工程技术人员观察测量、测试数据的变化及其趋势，从而使得虚拟仪器的前面板变得更加形象和直观。

除了提供丰富的数值显示控件、字符串显示控件、布尔按钮显示控件支持开发者设计良好的虚拟仪器人机交互界面，LabVIEW 还提供了强大的数据显示功能支持，程序设计人员通过简单的配置和编程的基本概念和技巧，就能够根据设计需求，定制出不同的显示界面。

新建 VI 中，右击程序框图，在控件选板中选择“函数→控件→新式→图形”，即可查看 LabVIEW 专业版所支持的图形显示控件，如图 3-1 所示。

图 3-1 LabVIEW 专业版所支持的图形显示控件

总体上看，这些显示控件分为图形类和图表类 2 种，其区别仅在于显示和更新的方式。其中图表类控件以追加显示数据点的模式，形成实时数据滚动刷新显示，类似于传统的示波器、波形记录仪等实体设备的显示功能。图形类显示控件一般直接显示若干数据点的分布状态，多用于历史数据的显示。

3.1.2　波形图的应用

在控件选板中选择“函数→控件→新式→图形→波形图”，可在程序前面板中插入波形图控件，其外观如图 3-2 所示。

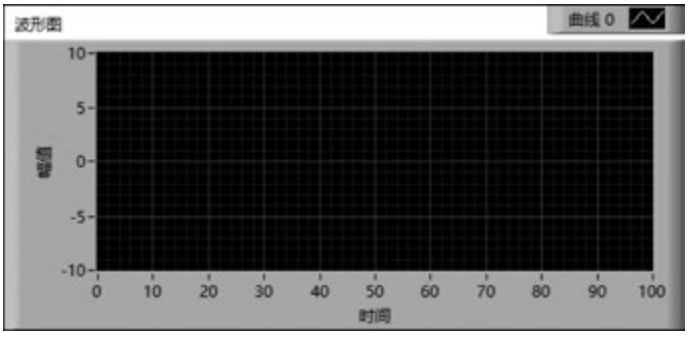


图 3-2　波形图控件外观

右击波形图控件，在弹出的快捷菜单中，可以进一步对波形图的各类属性进行操作。常用属性设置界面如图 3-3 所示。

图 3-3　常用属性设置界面

波形图控件使用方式灵活多样，主要分为以下三个场景。

（1）单个参数采集数据的显示。这里以随机数产生方式模拟单路数据采集，每间隔5ms采集一次数据，采集100个数据后，以波形图方式显示。100点单路数据采集及波形图显示前面板与程序框图如图3-4所示。

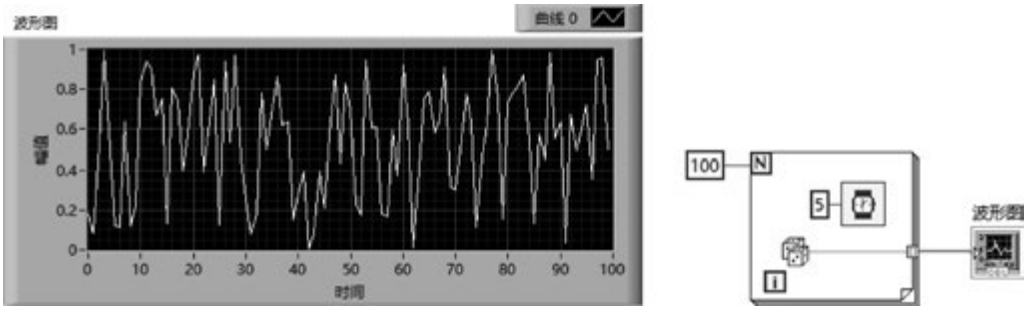


图 3-4　100点单路数据采集及波形图显示前面板与程序框图

（2）多路参数的数组方式显示。将采集的多路数据进一步封装为二维数组，作为波形图显示控件的数据源，即可实现多路参数同屏显示功能。多路参数的数组方式显示前面板与程序框图如图3-5所示。

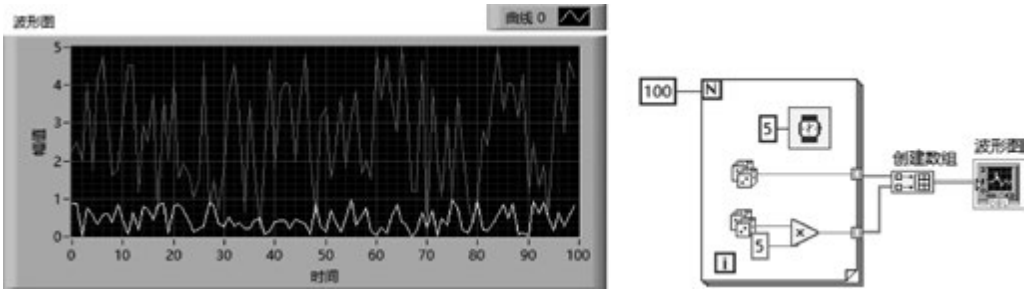


图 3-5　多路参数的数组方式显示前面板与程序框图

（3）多路参数的簇数组显示。将采集的多路数据首先封装为簇类型数据，然后将簇数据封装为数组，作为波形图显示控件的数据源。多路参数的簇数组显示前面板与程序框图如图3-6所示。

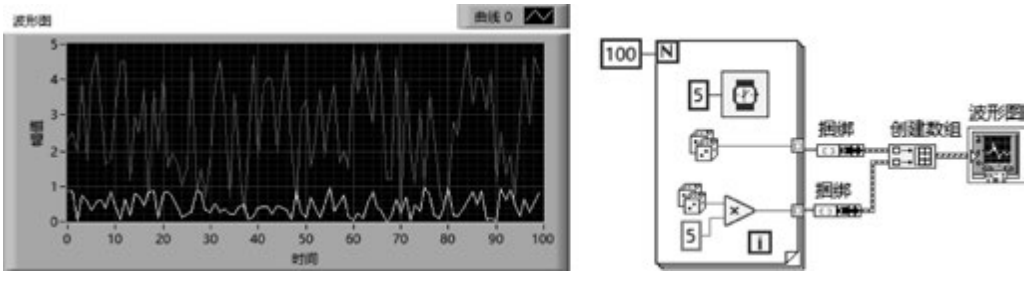


图 3-6　多路参数的簇数组显示前面板与程序框图

3.1.3　波形图表的应用

在控件选板中选择“函数→控件→新式→图形→波形图表”，可在程序前面板中插入波形图表控件，其外观如图3-7所示。与波形图不同的是，波形图表默认不显示坐标图背景。

图 3-7　波形图表控件外观

右击波形图表控件，在弹出的快捷菜单中，可以进一步对波形图表的各类属性进行操作。波形图表显示控件的常用属性设置界面与波形图相仿。

波形图表控件使用方式灵活多样，主要分为以下两个场景。

（1）单曲线波形图表显示。与波形图显示控件不同的是，波形图表显示控件一般置于循环内部，用以实时显示采集的数据波形。这里同样以随机数产生方式模拟单路数据采集，设定采样间隔为 200ms，对应的单曲线波形实时显示前面板与程序框图如图 3-8 所示。

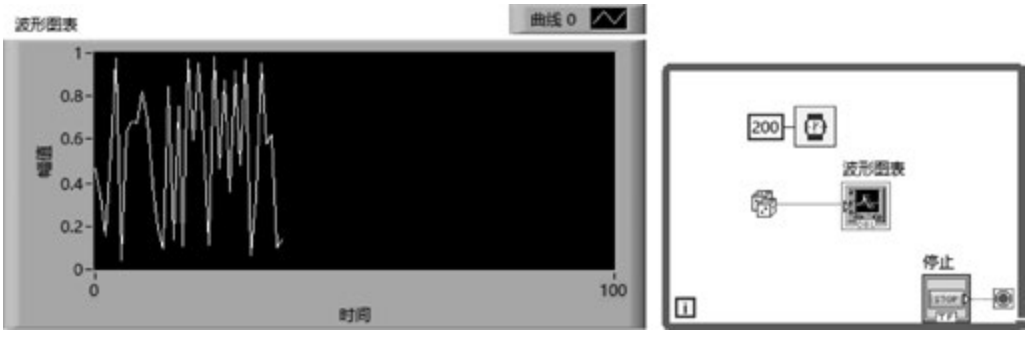


图 3-8　单曲线波形实时显示前面板与程序框图

（2）簇数据类型多曲线波形图表显示。当需要在波形图表中显示多曲线时，首先将实时采集的每个数据封装为簇数据，然后将簇数据作为波形图表显示控件的数据源，簇数据类型多曲线波形图表显示前面板和程序框图如图 3-9 所示。

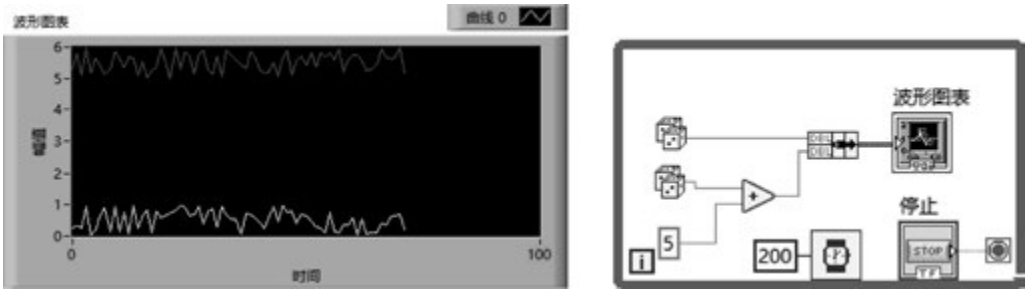


图 3-9　簇数据类型多曲线波形图表显示前面板和程序框图

3.1.4　XY 图的应用

控件选板中选择“函数→控件→新式→图形→XY 图”，可在程序前面板中插入 XY 图

控件，其外观如图 3-10 所示。

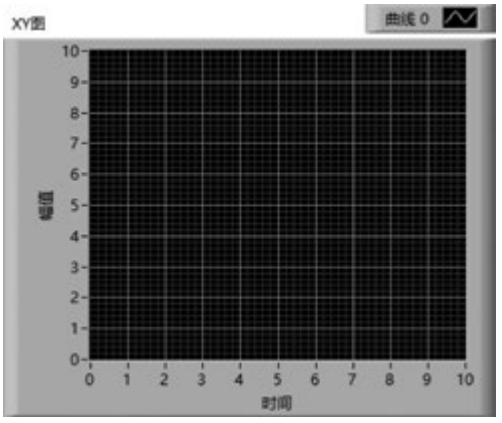


图 3-10　XY 图显示控件外观

XY 图显示一般多用于两个相互依赖的变量取值的分布状态。与波形图显示控件类似，该控件也是一次性完整绘制数据波形，其使用方式灵活多样，主要分为以下两个场景。

（1）单曲线 XY 图显示。在 XY 图显示控件中绘制单曲线，有两种方式。一是将 2 路数据采集结果所得 X 数组、Y 数组捆绑为簇数据，作为 XY 图显示控件的数据源；二是在采集 2 路数据的过程中，每次采集的数据都首先捆绑为簇数据，最后生成簇数组，作为 XY 图显示控件的数据源。这里调用“基本函数发生器”（函数→信号处理→波形生成→基本函数发生器），产生 2 路波形数据，调用“获取波形成分”（函数→编程→波形→获取波形成分），提取波形数据中的一维数组数据，然后将 2 路波形信号所获取的一维数组封装为簇数据，作为 XY 图显示控件的数据源。对应的单曲线 XY 图显示前面板和程序框图如图 3-11 所示。

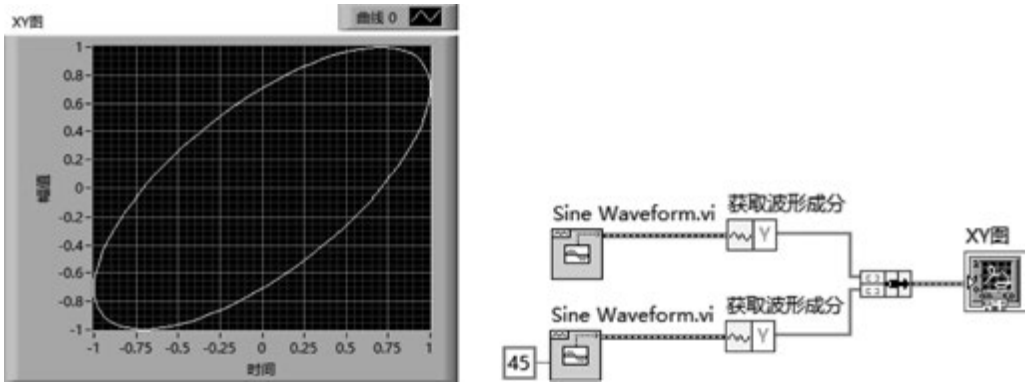


图 3-11　单曲线 XY 图显示前面板和程序框图

（2）多曲线 XY 图显示。多曲线 XY 图是在单曲线 XY 图显示基础上，将每个单曲线 XY 图显示数据封装为数组，将数组作为 XY 图显示控件的数据源。对应的多曲线 XY 图显示前面板和程序框图示例如图 3-12 所示。

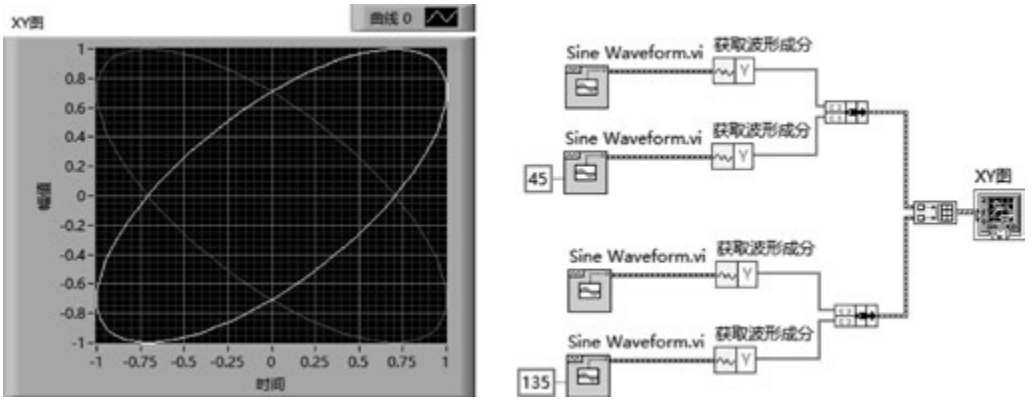


图 3-12　多曲线 XY 图显示前面板和程序框图示例

3.2　子 VI 的封装和调用

本节简要介绍 LabVIEW 程序设计中子 VI 的基本概念、创建过程以及子 VI 的调用方法，并给出多态子 VI 的基本概念、创建过程以及多态子 VI 的调用方法。

3.2.1　子 VI 设计的一般步骤

子 VI 相当于字符式编程环境下的函数或者子程序，是实现代码复用、程序模块化的重要手段。对于 LabVIEW 这种图形化编程环境，子 VI 还有大幅度减小代码占用的屏幕面积，增强程序可读性的重要作用。

创建子 VI 的方法有二，一为创建新 VI 实现，二为提取现有代码部分内容封装为子 VI。第一种方法在初学阶段比较常用，第二种方法在复杂程序设计中比较常用。LabVIEW 中按照第一种方法进行子 VI 的创建，分为“创建 VI”“编辑子 VI 图标”“建立连线器端子”三大步骤。

(1) 创建 VI。如同普通 VI 编写，完成期望功能的前面板设计、程序框图设计，形成一个完整可运行的 VI。这里以计算长方形面积为例(已知长方形的长和宽)，说明 VI 创建过程。

LabVIEW 开发环境下，单击“文件→新建 VI”，新建一个空白 VI。

右击前面板，打开控件选板，选择“控件→新式→数值”，添加 2 个数值输入控件、1 个数值显示控件，调整大小及其显示位置，子 VI 前面板设计结果如图 3-13 所示。

右击程序框图，打开函数选板，选择“函数→编程→数值→乘”，添加乘法计算节点，子 VI 程序连线如图 3-14 所示。

图 3-13　子 VI 前面板设计结果

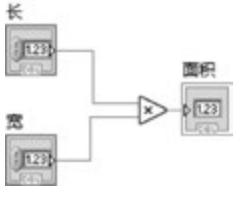


图 3-14　子 VI 程序连线

指定子 VI 程序文件存储的路径和名称，完成子 VI 文件存储。

（2）编辑子 VI 图标。构建子 VI 独特的图标，这一步往往可以省略，只不过会导致子 VI 在程序框图中的可辨识度下降。

右击（或双击）VI 右上角图标，在弹出的快捷菜单中选择“编辑图标”，弹出如图 3-15 所示的“图标编辑器”对话框。该对话框中可以设计自定义的子 VI 图标，使得子 VI 在调用过程中具有更好的可辨识度。

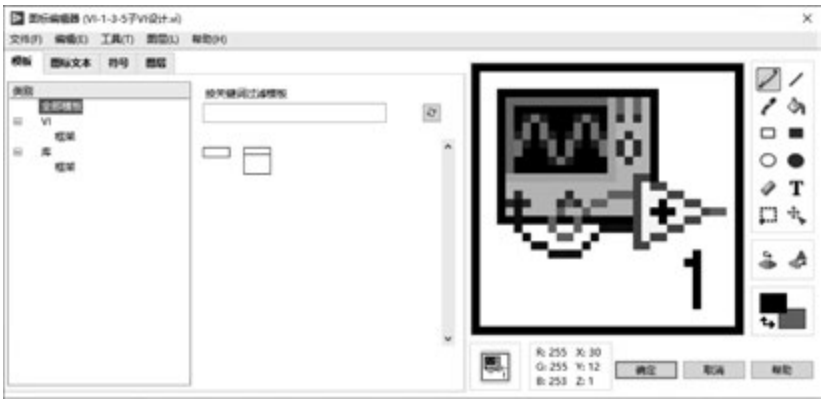


图 3-15 “图标编辑器”对话框

图标编辑器使用比较简单，这里不再赘述，读者可以自行探索。

（3）建立连线器端子。定义子 VI 至关重要的一步是确定子 VI 输入输出端口数量，并将每个端口与前面板控件对象关联。右击 VI 右上角图标，在弹出的快捷菜单中选择“模式”，显示 LabVIEW 提供的接线端子模板。由于长方形面积计算属于 2 输入 1 输出类型的接线端子，所以选择如图 3-16 所示的连接器模板。

选择完成后，子 VI 连接端口设置结果如图 3-17 所示。

图 3-16 选择连接器模板

图 3-17 子 VI 连接端口设置结果

单击连接器连线端口，选择与该端口关联的前面板控件，即建立起 2 个输入端口与前面板中数值输入控件（长和宽）之间的关联关系，然后建立连接器输出端口与前面板中数值显示控件之间的关联关系，用以表征长方形面积计算结果，子 VI 设计完成后的端口连接器和图标如图 3-18 所示。

经过上述三个步骤，对子 VI 进行命名、保存，才算是最终完成子 VI 的创建。

图 3-18　子 VI 设计完成后的端口连接器和图标

3.2.2　子 VI 的调用

在新建的 VI 中调用子 VI，方法比较简单。右击程序框图，选择“选择 VI”，弹出子 VI 调用文件对话框，如图 3-19 所示。

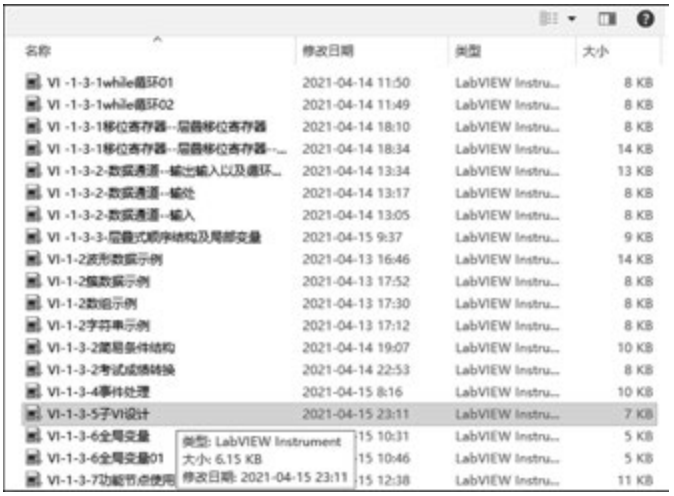


图 3-19　子 VI 调用文件对话框

选择上例中设计的子 VI，单击对话框中“确定”按钮，完成计算长方形面积的子 VI 调用。新建的 VI 中创建 2 个数值输入控件，修改标签分别为“长”和“宽”，创建数值显示控件，修改标签为“面积”，与调用的子 VI 连线，对应的子 VI 程序框图如图 3-20 所示。

单击 LabVIEW 开发环境中工具栏图标  ，选择连续运行。当输入长方形的长、宽参数后，调用子 VI 程序计算长方形面积，如图 3-21 所示。

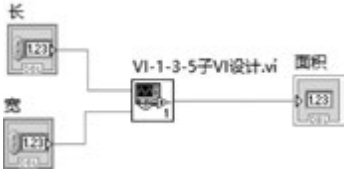


图 3-20　子 VI 程序框图

图 3-21　调用子 VI 程序计算长方形面积

微课视频

3.2.3 多态子 VI 的设计

1. 多态 VI 的基本概念

多态属于面向对象程序设计的核心概念，可以简单地概括为“一个接口，多种方法”，即同一个函数名称，但是可以接收多种参数类型的函数，实现不同的功能。多态函数的出现使得程序的代码更加灵活、可扩展性更强、代码更加易于维护。LabVIEW 中的多态 VI 是具有相同模式连线板的子 VI 的集合。多态 VI 集合中的每个 VI 都是多态 VI 的一个实例，每个实例都有至少一个输入或输出接线端接收的数据类型与其他实例不同。当程序设计过程中需要对不同数据类型执行相同的操作，或必须在不同的数据类型之间实现不同的操作时，可创建多态 VI。

2. 多态 VI 的创建方法

LabVIEW 中多态 VI 本质上是多个 VI 程序的集合，不需要进行程序框图设计，只需要集成事先创建好的 VI 即可，具体步骤如下。

(1) 新建多态 VI。启动 LabVIEW，开发环境菜单栏选择“文件→新建(N)”，在弹出的对话框中选择“多态 VI”，单击“确定”按钮，创建一个空的多态 VI。

(2) 多态 VI 编辑界面中，单击“添加”按钮，依次添加事先创建的接口模式相同、参数类型不同的 VI。

(3) 可根据需要进一步单击“编辑图标”按钮，进行自定义多态 VI 的图标设计，其他设置可选择默认状态。

(4) 保存多态 VI。

为了说明上述步骤，假设需要完成名为 myAdd 的多态 VI 设计，实现两个输入参数 a、b 的加法运算。输入参数 a、b 可以是以下三种情况。

(1) a、b 均为整数类型，输出结果为整数，其值为 a+b。

(2) a、b 均为浮点类型数组，输出结果为数组，其值为数组中对应位置数据元素的加法运算结果。

(3) a、b 均为布尔类型，输出结果为布尔，其值为 a、b 的或运算结果。

首先分别编写三个子 VI，对应三种输入类型，子 VI 分别命名为“Add_Int. vi”“Add_Array. vi”“Add_Bool. vi”。

“Add_Int. vi”前面板和程序框图设计如图 3-22 所示。

“Add_ Array. vi”前面板和程序框图设计如图 3-23 所示。

图 3-22 “Add_Int. vi”前面板和程序框图设计

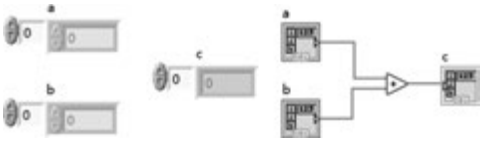


图 3-23 “Add_ Array. vi”前面板和程序框图设计

“Add_ Bool. vi”前面板和程序框图设计如图 3-24 所示。

完成三个子 VI 设计后，LabVIEW 菜单栏选择“文件→新建→VI→多态 VI”，打开新建多态 VI 对话框，在如图 3-25 所示的多态 VI 编辑界面中依次添加新建的三个子 VI，并编辑多态 VI 图标，勾选“默认显示选择器”复选框(可以提供多态 VI 调用时更好的可读性)。

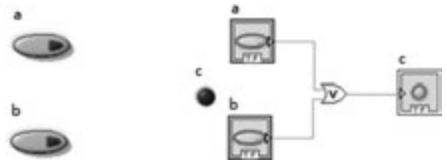


图 3-24　“Add_ Bool. vi”前面板和程序框图设计

图 3-25　多态 VI 编辑界面

完成编辑后，在多态 VI 编辑界面菜单栏选择“文件另存为(A)”，以 myAdd 为名称，保存设计的多态 VI，结束多态 VI 创建。

3. 多态 VI 的调用

多态 VI 程序作为子程序被调用时，可以自动适应所连线的输入数据类型，自动选择实际处理的 VI 程序并得到正确的运算结果。

启动 LabVIEW，新建 VI，命名为 TestVI，右击程序框图，选择“选择 VI”，调用 myAdd.vi。多态 VI 调用测试程序的前面板和程序框图的设计结果如图 3-26 所示。

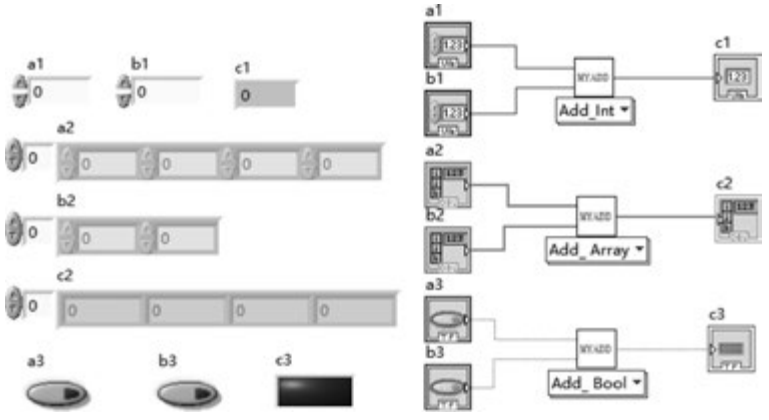


图 3-26　多态 VI 调用测试程序的前面板和程序框图的设计结果

3.3 文件操作

本节简要介绍 LabVIEW 程序设计中文件操作的基本概念、文件操作常用的函数节点以及文件操作相关程序的一般结构，并结合应用实例给出文件写入、文件读取两类应用的实现方法。

3.3.1 文件 I/O 的基本概念

在很多应用程序特别是数据采集应用程序设计开发过程中，往往需要将采集的数据以一定的文件格式存储在计算机磁盘。比如测试测量系统中采集数据以文本文件、表格文件或者 LabVIEW 专有的 TDM 文件保存，声音数据以 WAV 文件格式存储，图像数据以 BMP、PNG 或 JPG 等文件格式存储，视频数据以 AVI 文件格式存储，以备未来进一步深入分析。文件操作在应用程序开发中具有十分重要的实践意义。

文件操作需要熟悉以下几个基本概念。

(1) 文件路径。文件路径是指用于描述文件在磁盘中存储位置的一组字符串。这组字符串通常由多个文件夹名和一个文件名组成，每个文件夹名之间用“/”或“\”分隔，其中最后一个名称是文件名，其余的名称则是文件夹名。

(2) 绝对路径。绝对路径是指文件在计算机磁盘中的绝对位置。描述该位置的字符串是从盘符开始的路径，形如 C:\windows\system32\cmd.exe。使用绝对路径时无须考虑当前的工作目录。

(3) 相对路径。相对路径是指文件在磁盘中的存储位置是相对于一个参照位置而言的路径。相对路径使用的关键就是“..\”符号的使用，“..\”指的是当前路径的上一级目录，“..\..\”指的是上两级目录，以此类推。由于软件安装时一般都会遵照使用者的意愿设置程序安装路径，如果使用绝对路径，则有可能因为文件相关操作导致文件定位错误的问题。所以文件操作时，一般应该尽可能地使用相对路径，避免使用绝对路径。

(4) 文件引用句柄。LabVIEW 中的文件引用句柄类似于 C 语言中的指针，是一种标识符，用以区分文件。打开一个文件时，会生成一个指向该文件引用的操作句柄，后续对于文件的所有操作都可以通过这个句柄调用相关节点得以实现。文件引用句柄中包含文件的位置、大小、读写权限等信息。

3.3.2 文件操作的主要函数节点

LabVIEW 中支持的文件类型众多，右击程序框图空白处，选择“函数→编程→文件 I/O”，可以查看文件 I/O 相关函数节点，如图 3-27 所示。

文件 I/O 函数子选板中提供了文本文件、电子表格文件、二进制文件、TDM 文件(LabVIEW 专有测量文件)等不同文件格式的技术支持，同时提供了不同文件格式的专有操作函数节点，使得 LabVIEW 的文件操作相比于字符式编程语言要简单很多。

右击程序框图空白处，选择“函数→编程→图像与声音→声音→文件”，可以查看声音文件操作相关函数节点，如图 3-28 所示。

图 3-27　文件 I/O 相关函数节点

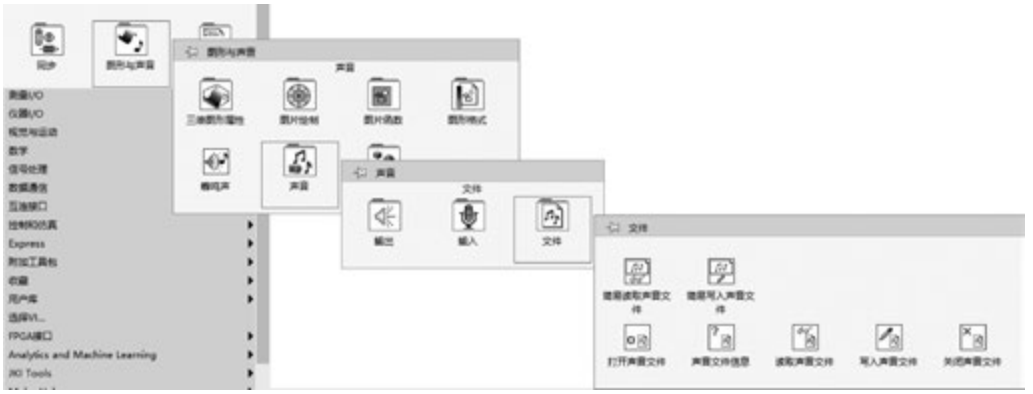


图 3-28　声音文件操作相关函数节点

右击程序框图空白处，选择“函数→视觉与运动→Vision Utilities→Files”，可以查看图像文件操作相关函数节点，如图 3-29 所示。

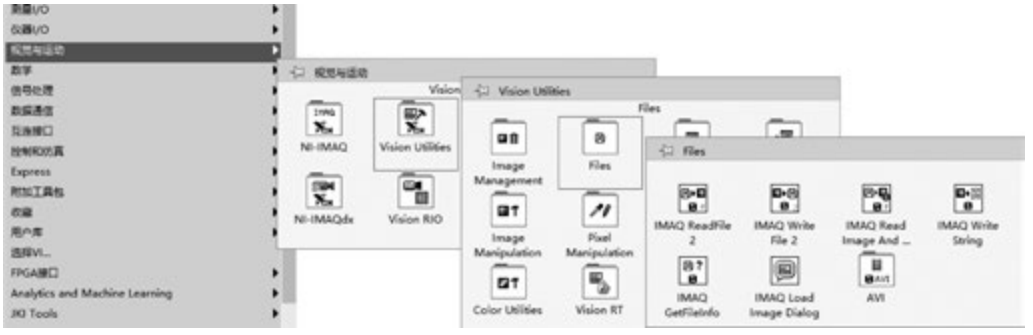


图 3-29　图像文件操作相关函数节点

无论哪种文件格式，对文件的操作一般包括 4 类函数节点的调用——打开/创建文件、文件读/写、关闭文件以及文件操作过程的错误处理。文件操作相关程序的基本结构如图 3-30 所示。

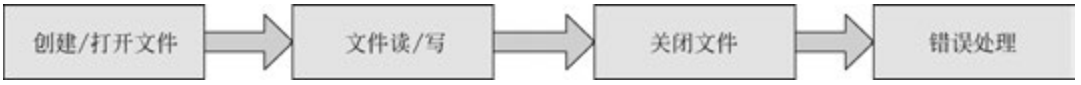


图 3-30　文件操作相关程序的基本结构


微课视频

3.3.3　文件操作的应用

数据采集时一种比较常用的数据存储方案就是将采集数据写入文本文件。假设拟开发的应用程序需要在当前 VI 所在路径下，创建一个名为 TestData.txt 的文件，以随机数模拟数据采集，以系统当前时间为数据采集时间，将采集数据和时间数据以逗号间隔，生成一次需要存储的字符串。各次采集数据之间以回车换行符号间隔，总计 100 次数据采集信息，逐次写入 TestData.txt，实现采集数据的文件保存。对应的数据采集及文件存储程序如图 3-31 所示。

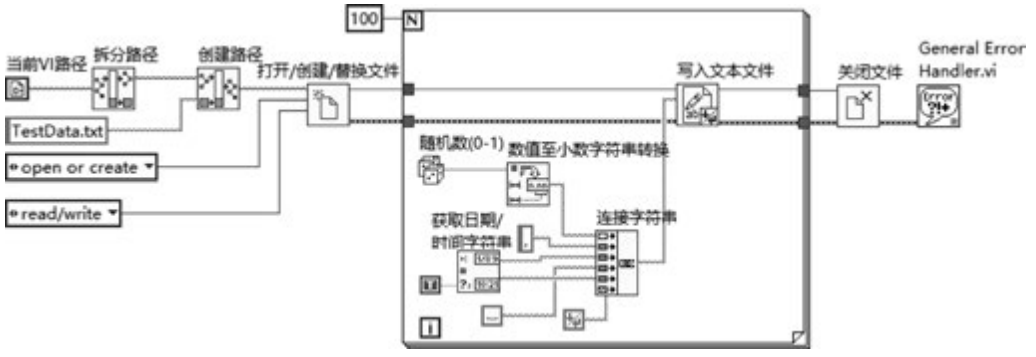


图 3-31　数据采集及文件存储程序

函数节点“打开/创建/替换文件”输出的“文件引用”(类似于 C 语言程序设计的指针，包含了文件操作必需的信息)连接函数节点“读取文本文件”，并设置读出字节数为－1(读出全部数据)，则可读出文本文件全部内容，对应的读取文本文件数据内容程序实现如图 3-32 所示。

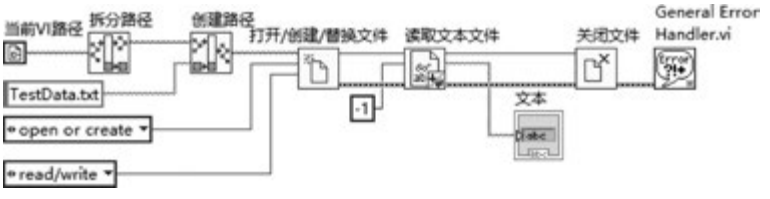


图 3-32　读取文本文件数据内容程序实现

运行程序，对应的前面板中字符串显示控件“文本”显示文件读取结果，如图 3-33 所示。

在人工智能相关应用程序开发过程中，经常要求采集数据以 CSV 格式的表格文件保存数据。LabVIEW 中提供了 CSV 格式文件操作的函数节点，极大地方便了这类文件的读写操作。

为了进一步说明文件类操作技巧，设计程序将采集的 2 路数据以及采集的时间信息以 CSV 表格文件存储。表格中第一列、第二列为不同区间的随机数模拟的数据采集结果，第三列为数据采集的时间。程序采集完毕数据后，判断当前 VI 路径下的子文件夹 FileTest 是否存在，如果不存在，则创建该文件夹下的 CSV 格式文件 TestData.csv，进而将采集的数据写入创建的 CSV 文件。将采集数据写入当前 VI 路径下的 CSV 文件程序框图如图 3-34 所示。

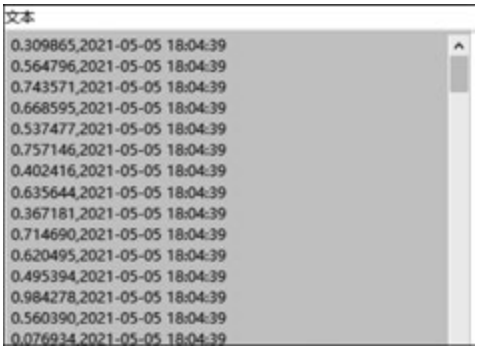


图 3-33　文件读取结果

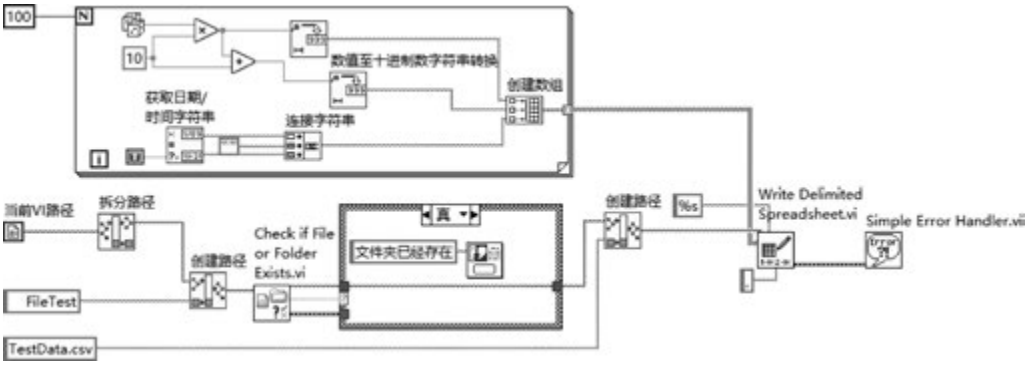


图 3-34　采集数据写入当前 VI 路径下的 CSV 文件程序框图

运行程序，并打开资源管理器，当前 VI 路径下子文件夹创建与 CSV 文件写入结果如图 3-35 所示。

图 3-35　当前 VI 路径下子文件夹创建与 CSV 文件写入结果

篇幅所限，其他类型文件的读写操作，读者可参阅 LabVIEW 相关范例，这里不再专门举例介绍。

3.4　联合编程

本节主要介绍 DLL、MATLAB、Python 等 3 种用以拓展 LabVIEW 程序功能的多平台联合编程技术，包括相关基本概念、程序创建以及 LabVIEW 程序设计中的调用方法。

3.4.1　基于 DLL 的联合编程技术

1. DLL 简介

DLL(Dynamic Linkable Library，动态链接库)可以理解为一种设计开发中可供使用的

功能仓库，它提供给开发者可以直接使用的变量、函数或类。只要遵循约定的 DLL 接口规范和调用方式，不同程序设计语言编写的 DLL 都可以相互调用。如 Windows 提供的系统 DLL(其中包括了 Windows 的 API)，在 Visual Basic、Visual C++、Visual C＃或 Delphi 等几乎任何传统开发环境中均能被调用。

DLL 的广泛使用有助于提高程序的模块化程度，使用 DLL 封装应用程序的有关功能，还能更加容易实现将软件系统各个功能部件的更新，而且能在不同编程语言中应用同一个 DLL 封装的功能，提高了代码的复用性。

作为一种图形化编程语言，LabVIEW 在测试测量领域功能极其强大，但是囿于图形化编程语言自身的局限性，在一些复杂算法、复杂功能的实现中，图形化编程语言往往会出现程序篇幅极大甚至难以实现的问题。

因此，复杂工程项目开发中发挥字符式编程语言(如 VB、VC、C＃等)和 LabVIEW 图形化编程语言的各自优势，比如借助字符式编程语言将复杂功能的实现封装为 DLL，而 LabVIEW 则继续专注于其测试测量的优势领域，通过调用字符式编程语言封装的 DLL，实现复杂问题的快速解决。这种方法对于加速 LabVIEW 程序设计进程、提升 LabVIEW 程序设计效率具有重要的实践意义。

DLL 是二进制文件，由于隐藏了源代码，项目交付时不必担心暴露源代码造成关键技术泄密，而且由于 DLL 文件相对独立存在，程序执行时 DLL 动态链接到应用程序中，因此一旦程序框架成熟稳定，则基于 DLL 的各部分功能均可通过替换 DLL 文件实现动态远程更新，这对于软件系统后期维护具有重要意义。

2. 创建 DLL

DLL 的创建可以使用多种字符式编程语言实现，比较常用的是基于 C++、C＃和 VB 等语言。LabVIEW 调用 DLL 时，则根据其实现语言的不同，调用方式有所差异。一般基于 C++编写的 DLL 使用“库与可执行程序”(函数→互连接口→库与可执行程序)相关函数节点，基于 C＃、VB 编写的 DLL 使用“. NET”(函数→互连接口→. NET)相关函数节点。下面给出 Visual Studio 开发平台下，基于 VB. NET 制作 DLL 的详细过程。

这里以文件的 Base64 编码、解码功能为例，制作本书后续开发实例中需要的 DLL。该 DLL 包含两个函数。

(1) Base64 编码函数，用以输出指定地址及名称的图像文件的 Base64 编码结果。

(2) Base64 解码函数，将 Base64 编码文件复原为对应的图像文件。

在 Visual Studio 2010 开发平台中单击“文件→新建→项目”，创建一个新项目，在“新建项目”对话框中，选择 Visual Basic，选择项目类型为“类库”，工程命名为“MyLib01”，VB. NET 新建 DLL 项目界面如图 3-36 所示。

单击“确定”按钮，在代码页编写图像文件，读取并编码为 Base64 字符串的函数，编写根据 Base64 字符串进行解码并将解码结果存储在指定文件的函数，VB. NET 中 DLL 函数设计相关代码如图 3-37 所示。

需要特别注意的是，函数前添加权限修饰符 public，否则该函数默认为私有访问权限，这会导致 LabVIEW 或者其他外部程序调用该动态链接库时，出现无权调用该函数报警提示信息。

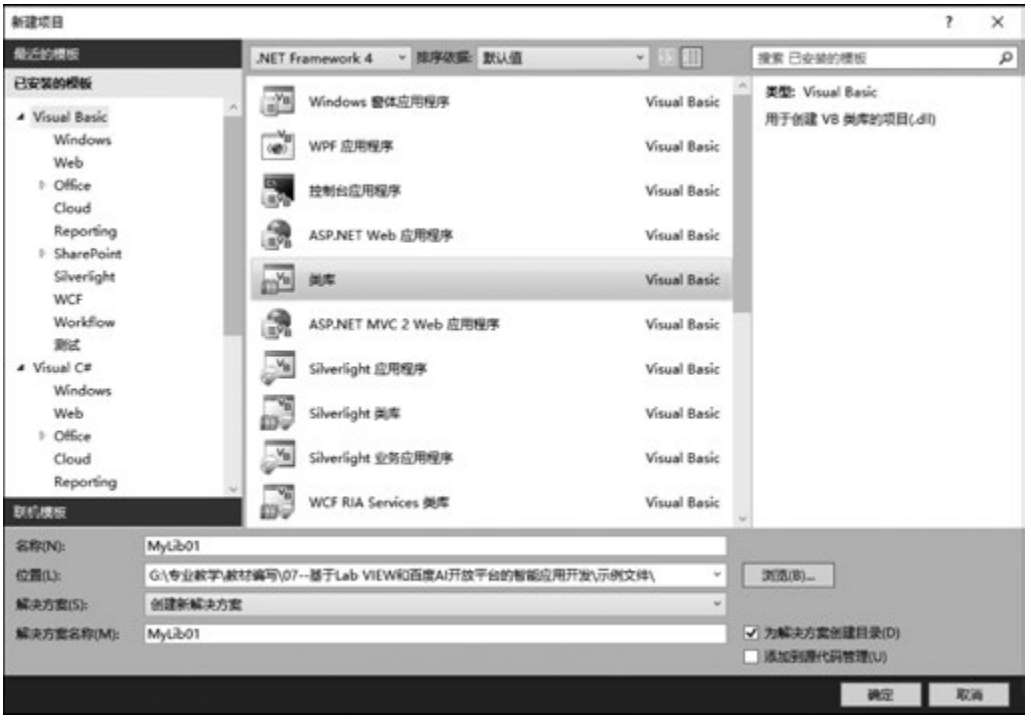


图 3-36　VB. NET 新建 DLL 项目界面

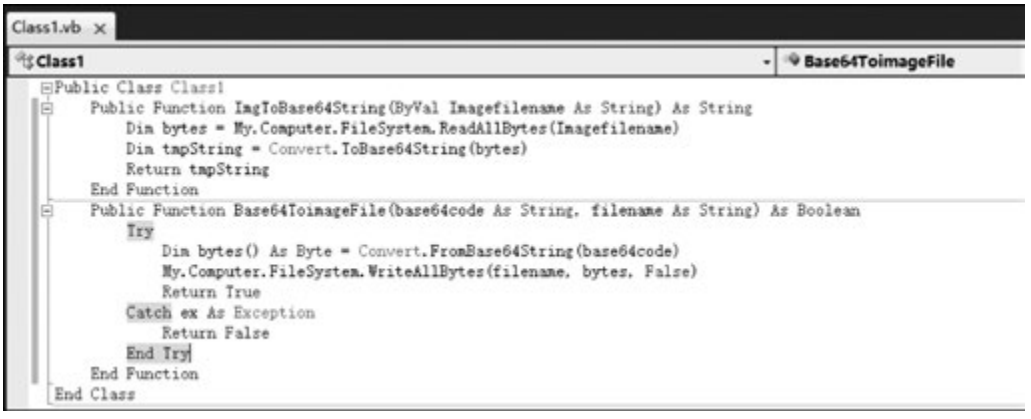


图 3-37　VB. NET 中 DLL 函数设计相关代码

完成功能代码编写后，单击开发环境菜单栏“生成→生成 MyLib01”，即可生成图像文件编解码动态链接库，如图 3-38 所示。

打开 Visual Studio 工程项目编译结果所在目录，即可查看编译生成的 DLL 文件，如图 3-39 所示。

3. 调用 DLL

在 LabVIEW 中对前一步生成的动态链接库进行调用，按照如下步骤完成。

新建 VI，右击程序框图空白处，在函数选板中选择“函数→互连接口→. NET→构造器节点”，添加函数节点“构造器节点”，如图 3-40 所示。

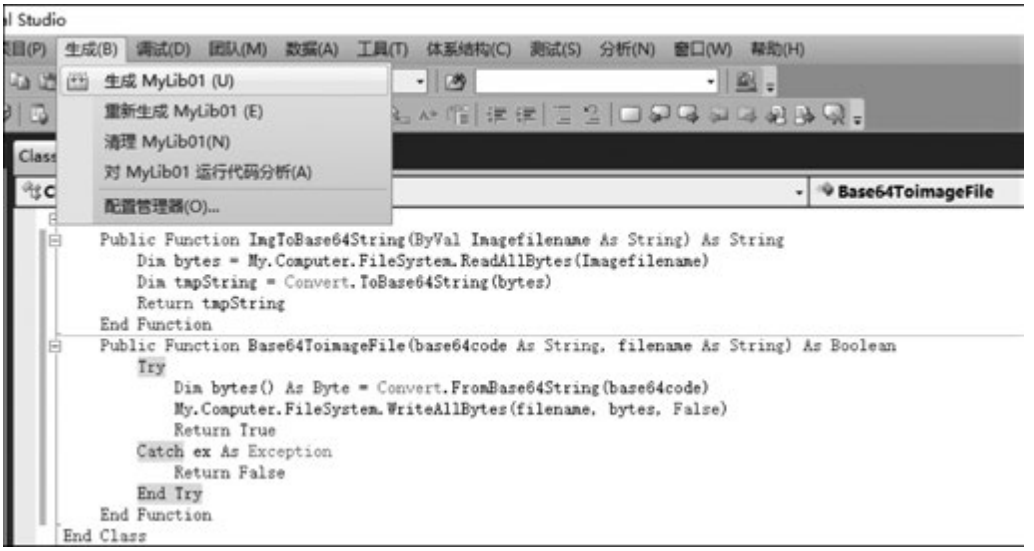


图 3-38　生成图像文件编解码动态链接库

图 3-39　查看编译生成的 DLL 文件

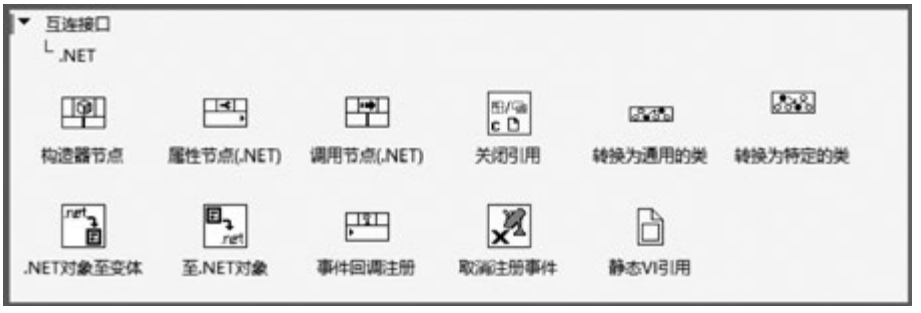


图 3-40　添加函数节点“构造器节点”

双击“构造器节点”图标，显示选择. NET 构造器对话框，如图 3-41 所示。

单击“浏览”按钮，在文件对话框中定位编译好的 DLL 文件，单击对象中“Class1”类，单击“确定”按钮，完成程序框图中. NET 对象的创建。

选择“函数→互连接口→. NET→调用节点”，添加函数节点“调用节点”，如图 3-42 所示。

单击函数节点“调用节点”中“方法”选项，选择 ImgToBase64String，如图 3-43 所示。

继续添加函数“调用节点”，设置其调用函数为 Base64ToimageFile，如图 3-44 所示。

图 3-41　选择.NET 构造器对话框

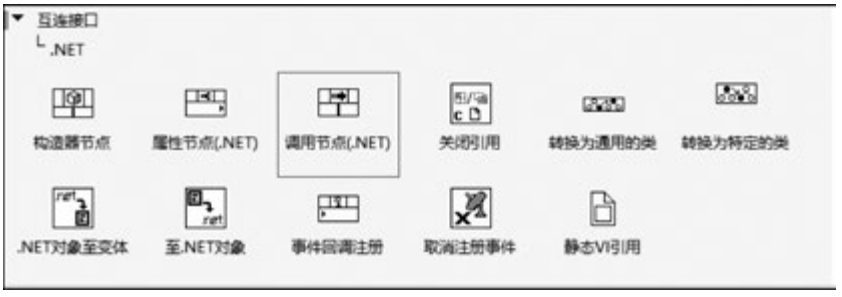


图 3-42　添加函数节点“调用节点”

图 3-43　选择 ImgToBase64String

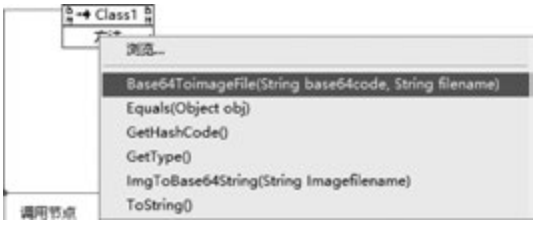


图 3-44　设置调用函数为 Base64ToimageFile

指定“调用节点”所需要的输入参数，设定“调用节点”输出数据的显示方式，即可完成基于.NET 构造器的图像文件编解码 DLL 的调用。为了使得 LabVIEW 程序具有更好的观测效果，前面板中添加字符串显示控件、图像显示控件（控件→Vision →Image Display Classic，需要安装计算机视觉 VDM 工具包），设计基于 DLL 的图像编解码程序前面板如图 3-45 所示。

对应地，基于 DLL 的图像编解码程序框图如图 3-46 所示。

程序执行时，打开图像文件对话框，选择拟进行编码的图像文件，获取文件地址及名称，将其作为 DLL 中函数 ImgToBase64String 的输入参数，编码结果（Base64 字符串）则作为

图 3-45　基于 DLL 的图像编解码程序前面板

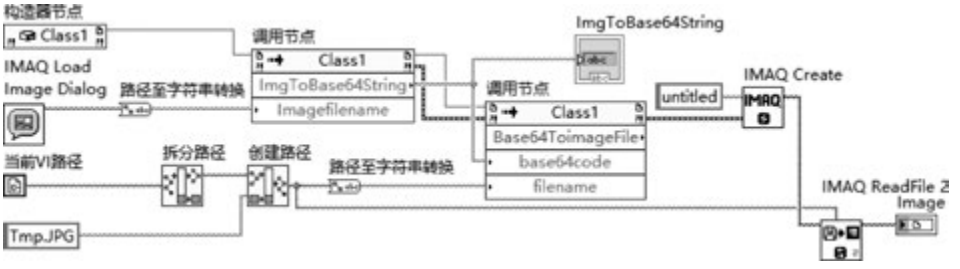


图 3-46　基于 DLL 的图像编解码程序框图

DLL 中函数 Base64ToimageFile 的输入参数 base64code 的取值，同时程序以当前 VI 所在路径构造指定图像文件地址及名称，作为 DLL 中函数 Base64ToimageFile 的输入参数 filename 的取值。基于 DLL 的图像文件 Base64 编码以及解码重建结果如图 3-47 所示。

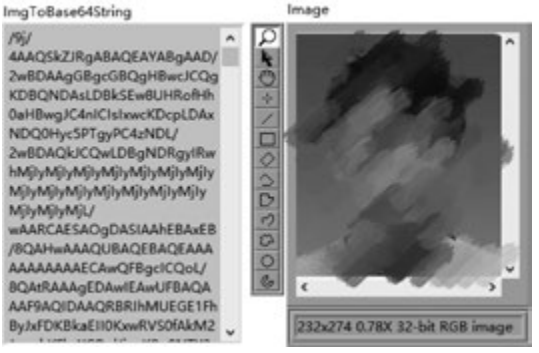


图 3-47　基于 DLL 的图像文件 Base64 编码以及解码重建结果

程序设计及运行结果可见，使用 VB.NET 开发语言仅需 3～5 行代码即可完成图像文件转换为 Base64 字符串功能。将其封装为 DLL，在 LabVIEW 中借助 DLL 技术，可极容易地实现复杂的程序功能，从而弥补 LabVIEW 开发平台的不足，拓展 LabVIEW 的应用场景，提高应用程序开发效率。


微课视频

3.4.2　基于 MATLAB 的联合编程技术

1. MATLAB 简介

MATLAB 是美国 MathWorks 公司开发的大型数学计算软件，目前用于数据分析、无

线通信、深度学习、图像处理与计算机视觉、信号处理、量化金融与风险管理、机器人、控制系统等领域。MATLAB 以其强大的功能和良好的应用生态，广泛地应用于科学研究和工程技术的各个领域。

MATLAB 提供了门类丰富且功能强大的模块集和工具箱。典型工具箱覆盖数据采集、神经网络、小波分析、信号处理、图像处理、概率统计、样条拟合、优化算法、偏微分方程求解、各类控制系统设计、实时快速原型及半物理仿真、嵌入式系统开发等，可以极大地加速科研人员创新研究、技术开发的进程。

MATLAB 还提供了与其他高级编程语言进行交互的函数库。该函数库的函数基于 DLL、ActiveX、DDE 等技术可以实现 MATLAB 与其他程序设计语言的混合编程。混合编程模式可以发挥不同计算机编程语言的优势，因而可以大幅提升设计开发的效率。

2. 创建 MATLAB 代码

打开 MATLAB 软件，其操作界面分为 4 个操作区域，如图 3-48 所示。

图 3-48　MATLAB 操作界面

一是位于顶部的菜单与工具栏，分为主页、绘图、APP 三大选项卡。其中主页选项卡分为文件、变量、代码、SIMULINK、环境、资源 6 类操控对象。

二是当前文件夹，亦称工作路径，一般设置为当前程序文件保存的文件夹路径。

三是命令行窗口，用来输入并运行 MATLAB 语言编写的代码。

四是工作区，也称为工作空间，用来暂存、查看 MATLAB 程序运行结果。

MATLAB 程序编写可以直接在命令行窗口中输入简单指令直接运行，但是如果程序流程较为复杂，且需要经常重复输入或者运行时，程序设计多以脚本或者函数设计为主，即基于 m 文件进行程序设计。脚本和函数的程序文件均以.m 为文件后缀名。

单击 MATLAB 菜单栏“主页→新建→脚本”，或者使用快捷键 Ctrl＋N，命令行窗口自动创建后缀名为.m 的脚本文件，如图 3-49 所示。

图 3-49　MATLAB 中创建脚本文件

单击菜单栏“运行”按钮，弹出文件保存对话框，选择脚本文件存储的文件夹，确定脚本文件名称，如图 3-50 所示。

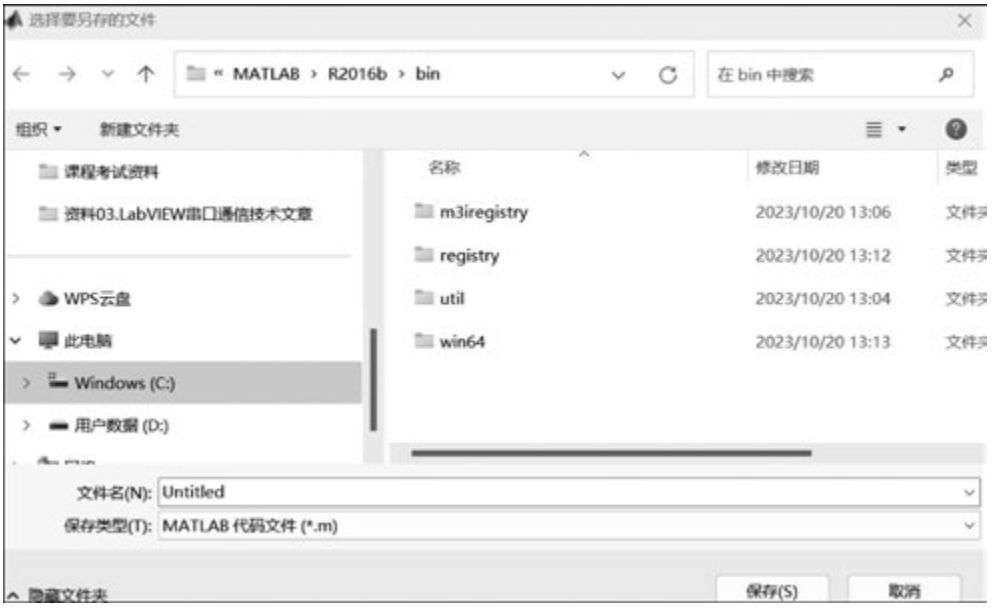


图 3-50　存储 MATLAB 脚本文件

需要注意的是，MATLAB 脚本文件名称必须以字母开头，并且仅包含字母、数字或下画线。单击 MATLAB 菜单栏“主页→新建→函数”，命令行窗口自动创建后缀名为. m 的函数文件。默认的函数声明格式如下。

```
function [ output_args ] = Untitled( input_args )
% UNTITLED 此处显示有关此函数的摘要
%   此处显示详细说明
end
```

函数声明由关键字 function 引导，由关键字 end 结束。函数声明中需要定义函数名（初次创建时默认函数名称为 Untitled1）、输出形参表和输入形参表。其中，函数名应当与文件名一致（保存函数文件时会默认以函数名作为文件名予以保存）。输入形参表如果包含多个参数，则以“,”为间隔，逐一定义每个输入参数。输出形参表默认处于“[]”内，也可以是以“,”为间隔的多个输出参数。函数体位于 function 声明语句和关键词 end 之间，函数体中对所有输入变量都要在代码中进行引用，所有输出变量都应该在代码中进行赋值。

如需设计函数对于传入的两路信号波形进行叠加运算，并对叠加结果进行 FFT，则可创建 MATLAB 函数实现 2 路信号叠加以及 FFT，如图 3-51 所示。

完成函数设计，既可以在命令行窗口中调用自定义的函数，也可以在脚本程序中调用自定义的函数。如果调用过程中出现无法识别自定义函数的问题，则需将自定义函数文件所在文件夹添加为 MATLAB 搜索路径之一。

3. 调用 MATLAB 代码

LabVIEW 在测试测量领域功能强大，但是在涉及复杂的测试数据分析处理方面还是略显不足，因而引入数学运算功能极其强大的 MATLAB，基于混合编程技术，可以大幅加

图 3-51　创建 MATLAB 函数实现 2 路信号叠加以及 FFT 变换

快虚拟仪器开发进程。

LabVIEW 提供了多种 MATLAB 混合编程的技术支持，包括基于 MATLAB 脚本节点、DDE、DLL、ActiveX 自动化等技术。其中基于 MATLAB 脚本节点的混合开发技术功能比较强大。在新建的 VI 中，右击程序框图空白处，在函数选板中选择“函数→数学→脚本与公式→脚本节点→MATLAB 脚本”，即可查看、获取 MATLAB 脚本节点，如图 3-52 所示。

图 3-52　查看、获取 MATLAB 脚本节点

在 MATLAB 脚本节点中，既可以直接调入已经存在的 MATLAB 脚本程序，也可以在 MATLAB 脚本节点中直接编写、调试 MATLAB 脚本程序。这使得开发者可以很方便地在自己的 LabVIEW 应用程序中使用 MATLAB 编写的算法和功能丰富的工具箱，能够极大地提高复杂应用程序的开发效率。但是这种开发模式无法脱离 MATLAB 环境，需要在计算机中安装有 MATLAB，而且对于输入、输出数据的类型有明确的要求，目前两者之间的数据通信仅支持 Real、Complex、字符串、路径以及 1D real、1D Complex、2D Real、2D Complex 8 种格式的数据。

为了进一步说明 LabVIEW 与 MATLAB 混合编程技术，这里通过 LabVIEW 产生 2 路不同频率的正弦波信号（带噪声），通过 MATLAB 脚本函数节点，调用 MATLAB 程序代码，对于 LabVIEW 中产生的 2 路带噪声正弦信号进行叠加运算，然后进行 FFT 变换，并显示叠加信号波形以及叠加信号的频谱分析结果。程序实现分为如下步骤。

（1）LabVIEW 中产生正弦波信号。新建 VI，右击程序框图空白处，在函数选板中选择“函数→Express→输入→仿真信号”，在弹出的仿真信号配置窗口中，20Hz 带噪声正弦信号参数配置如图 3-53 所示。

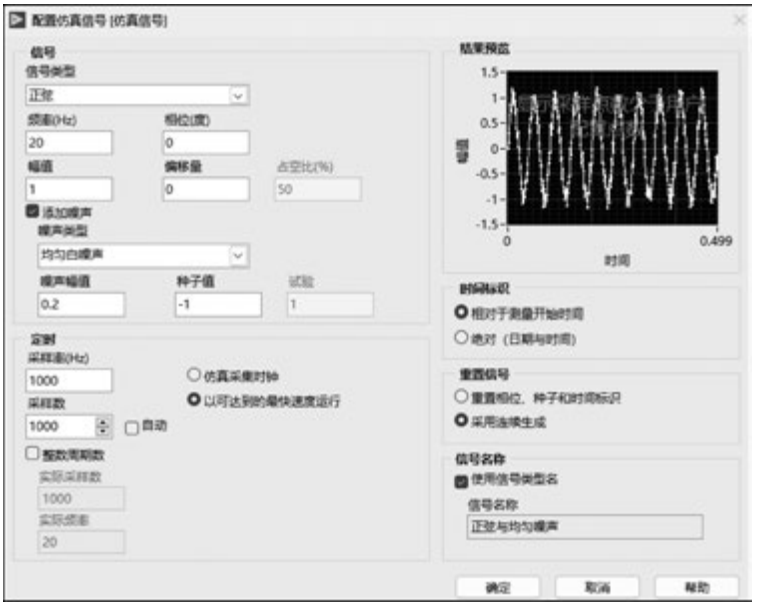


图 3-53　20Hz 带噪声正弦信号参数配置

同样方法，调用 ExpressVI“仿真信号”，完成 100Hz 带噪声正弦信号参数配置，如图 3-54 所示。

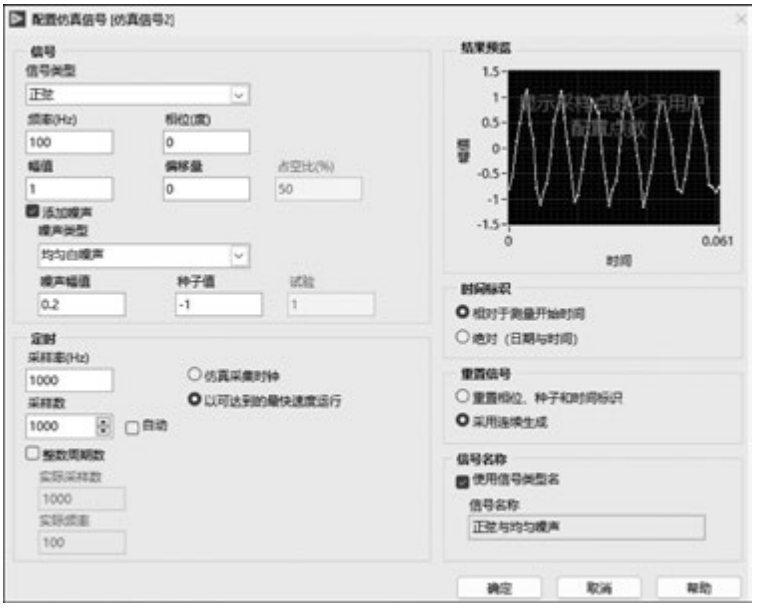


图 3-54　100Hz 带噪声正弦信号参数配置

（2）添加 MATLAB 脚本节点。右击程序框图空白处，在函数选板中选择“函数→数学→脚本与公式→脚本节点→MATLAB 脚本”，完成程序框图中 MATLAB 脚本节点的添加。右击 MATLAB 脚本节点边框，在弹出的快捷菜单中选择“导入”，可以直接载入事先编写好的 MATLAB 脚本代码。当然也可以在 MATLAB 脚本节点中直接编写脚本代码。

（3）配置 MATLAB 脚本节点输入、输出参数。右击 MATLAB 脚本节点边框，在弹出的快捷菜单中选择“添加输入”，为 MATLAB 脚本节点边框添加一个输入参数，命名为 s1。右击 MATLAB 脚本节点边框新添加的输入参数，在弹出的快捷菜单中选择“选择数据类型→1-D Array of Real”，定义 MATLAB 脚本节点输入参数的数据类型，如图 3-55 所示。

图 3-55　定义 MATLAB 脚本节点输入参数的数据类型

同样方法，完成输入参数 s2、输出参数 s 与 filt 的创建和数据类型定义。本例中，所有输入数据、输出数据的数据类型均为浮点型一维实数数组。

（4）转换数据类型。调用函数节点“从动态数据转换”（函数→Express→信号操作→从动态信号转换），将函数节点“仿真信号”生成的 20Hz 带噪声正弦波信号转换为一维数组，与 MATLAB 脚本节点输入参数 s1 连接。将函数节点“仿真信号”生成的 100Hz 带噪声正弦波信号转换为一维数组，与 MATLAB 脚本节点输入参数 s2 连接。

（5）MATLAB 处理数据。MATLAB 脚本节点中，首先对 LabVIEW 传入 s1、s2 的参数进行叠加运算，然后进行 FFT 变换，叠加信号以及 FFT 变换结果通过输出参数 s、filt 传递至 LabVIEW。注意输出参数 s、filt 必须与 MATLAB 脚本节点中有关代码保存计算结果的变量名称保持一致。MATLAB 脚本节点的输入输出参数以及脚本代码如图 3-56 所示。

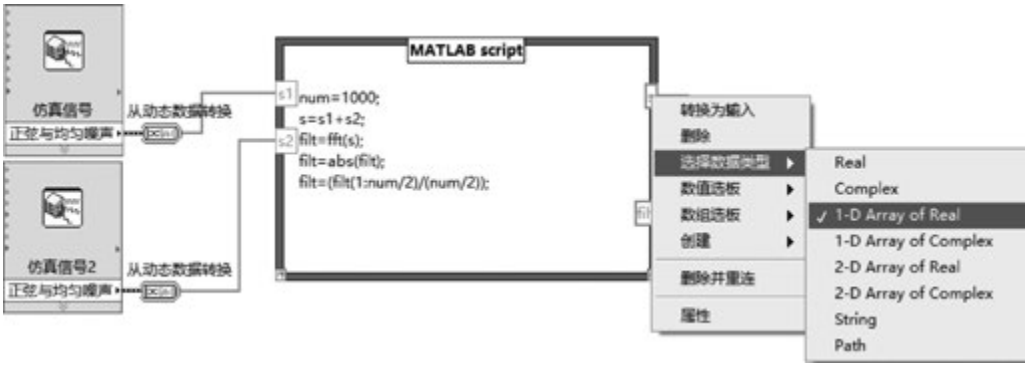


图 3-56　MATLAB 脚本节点的输入输出参数以及脚本代码

（6）MATLAB 脚本节点输出结果显示。进一步地，将 MATLAB 脚本节点输出参数 s 与名为“波形图”的控件连接，用以显示叠加后的信号波形。MATLAB 脚本节点输出参数 filt 与名为“频谱图”的控件连接，用以显示叠加信号的频谱分析结果。完整的基于 MATLAB 脚本节点调用实现信号叠加以及频谱分析的程序框图如图 3-57 所示。

上述 MATLAB 脚本节点的 MATLAB 脚本代码，既可以直接在节点中编写，又可以右击节点边框，在弹出的快捷菜单中选择“导入”，导入事先编辑好的 MATLAB 脚本代码。

运行程序，前面板显示叠加信号的波形以及频谱，如图 3-58 所示。带均匀白噪声的 50Hz 正弦波信号与带均匀白噪声的 100Hz 正弦波信号进行叠加运算后，由 FFT 变换结果可清晰判断叠加信号的频率成分。

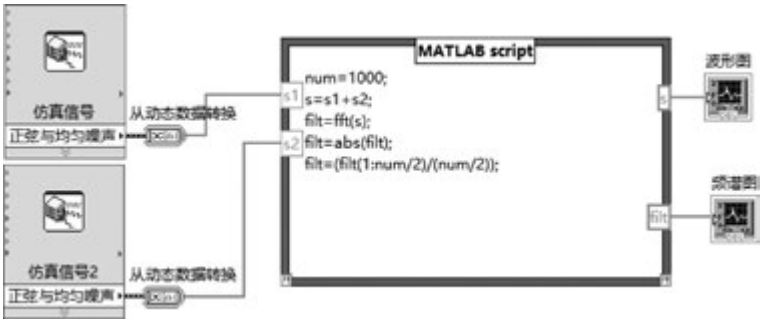


图 3-57　基于 MATLAB 脚本节点调用实现信号叠加以及频谱分析的程序框图

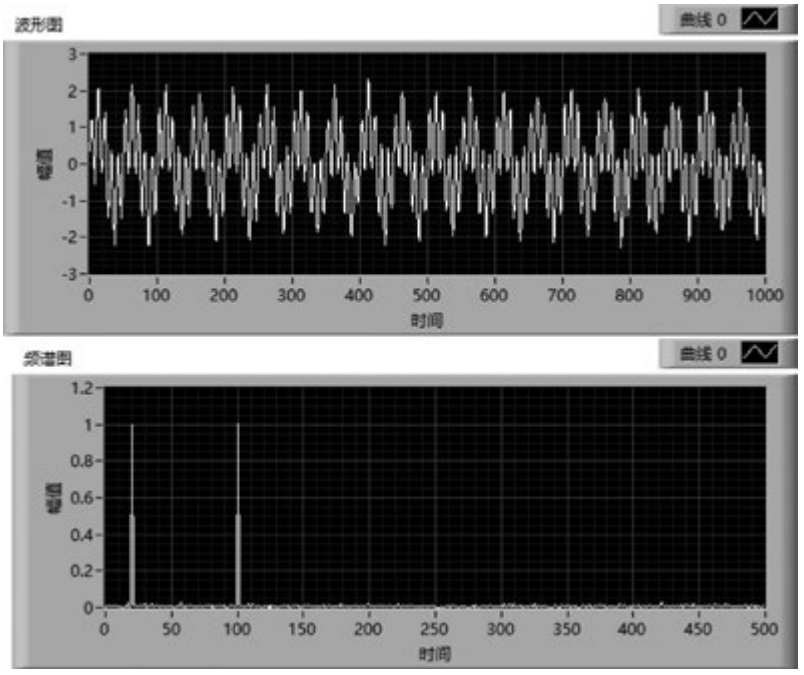


图 3-58　叠加信号的波形以及频谱

如果 MATLAB 中需要对 LabVIEW 应用程序中产生的数据进行复杂分析处理，则不可避免地涉及自定义的 MATLAB 函数调用。比如上述 MATLAB 脚本代码可以封装为如下自定义函数。

```
function[s,filt] = myfft_fun(s1,s2,num)
% 有关此函数的摘要
% s1 为第一路信号;s2 为第二路信号,num 为信号点数
% s 为叠加后信号,filt 为信号 s 的 FFT 结果
s = s1 + s2;
filt = fft(s);
filt = abs(filt);
filt = (filt(1:num/2)/(num/2));
end
```

MATLAB 脚本节点中可直接调用自定义函数，实现相关数据分析处理功能，如图 3-59 所示。

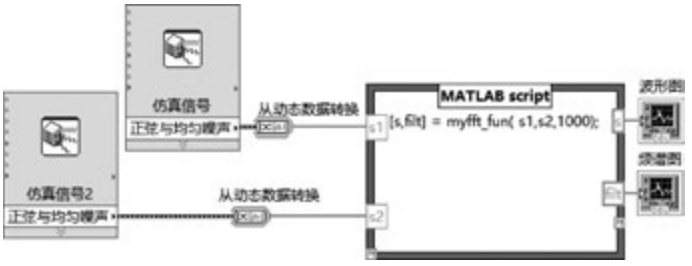


图 3-59　MATLAB 脚本节点调用自定义函数

但是 MATLAB 中自定义函数在 LabVIEW 提供的 MATLAB 脚本节点中调用时，往往会出现无法识别或者检测不到对应的 m 文件等问题。一种解决方案就是将测试无误的 m 函数文件复制粘贴至 MATLAB 默认工作路径之下，如图 3-60 所示。

图 3-60　将测试无误的 m 函数文件复制粘贴至 MATLAB 默认工作路径之下

另一种解决方案就是在 MATLAB 开发环境中，选择“主页→环境”，单击“设置路径”按钮，在弹出的对话框中，单击“添加文件夹”按钮，将自定义函数文件所在文件夹作为 MATLAB 搜索路径之一，如图 3-61 所示。

图 3-61　将自定义函数文件所在文件夹作为 MATLAB 搜索路径之一


微课视频

3.4.3 基于 Python 的联合编程技术

1. Python 简介

Python 是一门高级的、面向对象的、解释型的编程语言，由 Guido van Rossum 于 1991 年开发。Python 语言简洁、易学、易读、易写，具有很好的可移植性、可扩展性和可维护性。Python 语言不仅内置了庞大的标准库，还提供了丰富的第三方库如 NumPy、Pandas、Requests 等，可以帮助开发人员快速、高效地处理各类工作，近年来在科学计算、机器学习（Machine Learning，ML）、人工智能（Artificial Intelligence，AI）等诸多领域得到广泛应用。

安装 Python 环境，需要注意 Python 版本，目前 LabVIEW 支持 3.6～3.10 版本的 Python。Python 环境的安装文件可在官网进行下载，NI 建议选择使用 32 位、3.10 以内的版本的 Python。安装完毕 Python 环境后，可利用 pip 工具安装各类强大的第三方工具包。

2. 创建 Python 代码

Python 环境安装配置完成后，可通过命令行测试是否安装成功。使用快捷键 Win＋R 打开命令行，输入 cmd 指令并按 Enter 键，在“cmd 控制台”输入 Python，然后按 Enter 键。如果返回 Python 版本相关信息，表示 Python 安装成功。进一步键入代码 print("Hello Python")，按下 Enter 键，即可测试 Python 开发环境是否正常，如图 3-62 所示。

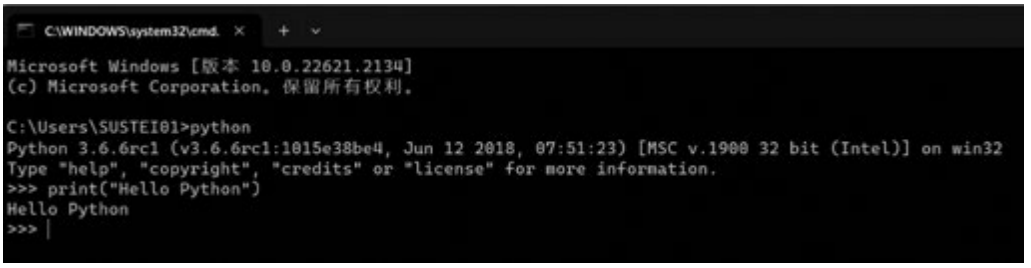


图 3-62　测试 Python 开发环境是否正常

Python 代码编辑器多种多样，既可以使用自带的 IDLE 编辑器，还可以使用 PyCharm、Visual Studio Code，甚至可以使用记事本进行程序编写。建议使用专业代码编辑器（如 PyCharm、Visual Studio Code）进行程序设计。可自行搜索文献了解相关软件环境的下载、安装和配置方法。

本例中使用 Visual Studio Code 给出判断指定年份值是否为闰年的 Python 函数代码，如图 3-63 所示。

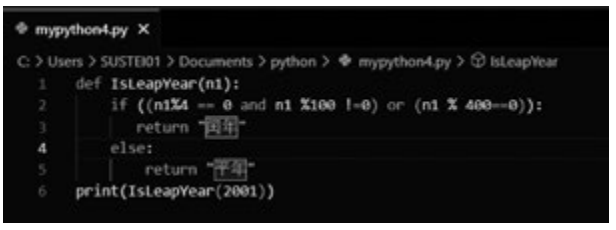


图 3-63　判断指定年份值是否为闰年的 Python 函数代码

图 3-63 中可见函数名为 IsLeapYear，输入参数为数值类型数据，输出参数字符串类型。最后一行代码对于函数功能进行测试。特别需要注意的是，Python 相关程序文件的存储路

径中不能存在中文字符，否则会导致后续 LabVIEW 中调用时出现错误。

3. 调用 Python 代码

LabVIEW 自从 2018 版开始支持在程序设计中直接调用 Python 应用程序，使得开发者能够有效利用 Python 提供的新算法和技术，充分发挥 LabVIEW 和 Python 各自优势，实现复杂应用程序的快速开发。

新建的 VI 中，右击程序框图空白处，函数选板中选择“函数→互连接口→Python”，即可查看调用 Python 的函数节点，如图 3-64 所示。

图 3-64　查看调用 Python 的函数节点

LabVIEW 中调用 Python 程序仅需要 3 个函数节点。“打开 Python 会话”用于打开并执行某一版本的 Python 所创建程序的会话。

“Python 节点”用来直接调用指定路径下的 Python 程序文件中的有关函数功能。该函数节点带有可扩展的输入输出端子，用以配置调用 Python 函数时传入对应的参数，或者输出需要的结果数据。可扩展的输入输出端子支持数值、数组、字符串、簇、布尔等数据类型，其中 LabVIEW 中的数组会被自动转换为 Python 中的 List 或者 NumPy，簇数据会被转换为 Tuples。

“关闭 Python 会话”用以释放打开 Python 会话所占用的系统资源。

为了进一步说明 LabVIEW & Python 联合编程技术，这里设计 LabVIEW 程序，对于前面板中用户输入的年份值进行是否闰年的判断。其中是否闰年的判断由 LabVIEW 调用 Python 函数实现。程序实现步骤如下。

(1) 设计程序前面板。根据程序设计意图，程序前面板中提供数值输入控件用以输入需要判断的年份值；提供布尔类型的确定按钮，用以触发判断是否闰年；提供字符串显示控件，用以显示判断结果；提供布尔类型停止按钮，用以结束程序运行。判断是否闰年程序前面板如图 3-65 所示。

图 3-65　判断是否闰年程序前面板

(2) 确定程序总体结构。程序设计采用 While 循环结构中内嵌事件结构的事件响应模式，实现程序运行过程中对于程序界面相关操作的连续响应，直至用户单击停止按钮结束程序运行。对应的事件响应模式程序框图如图 3-66 所示。

图 3-66　事件响应模式程序框图

（3）打开 Python 会话。在前面板“确定按钮”事件分支内，调用函数节点“打开 Python 会话”（函数→互连接口→Python→打开 Python 会话），右击其输入端口“Python 版本”，选择“创建→常量”，在生成的字符串常量中键入当前 Python 环境版本 3.6，完成函数节点调用 Python 代码的版本参数设置。

（4）调用 Python 函数。调用函数节点“Python 节点”（函数→互连接口→Python→Python 节点），右击其输入端口“模块路径”，选择“创建→常量”，在生成的路径常量中键入事先编写好的 Python 源代码文件路径；右击其输入端口“函数名称”，选择“创建→常量”，在生成的字符串常量中键入 Python 程序判断是否闰年的函数名称 IsLeapYear。

（5）设置函数节点“Python 节点”输入输出参数。函数节点“Python 节点”为可扩展函数。默认第一参数左侧为返回值类型，右侧为返回值；其他接口为输入参数，可以拖曳函数图标中操作句柄，更改输入参数的个数。添加“字符串常量”（函数→编程→字符串→字符串常量），连接“Python 节点”端口“返回类型”，设置返回值类型为字符串类型；前面板数值输入控件图标连接“Python 节点”端口“输入参数”，设置调用函数的输入参数类型和取值；“Python 节点”端口“返回值”连接前面板字符串显示控件图标，实现 Python 函数调用结果的显示功能。

（6）释放 Python 函数调用资源。调用函数节点“关闭 Python 会话”（函数→互连接口→Python→关闭 Python 会话），在 Python 函数调用结束后，释放其占用的系统资源。

最终完成的 LabVIEW 中调用 Python 函数程序框图如图 3-67 所示。

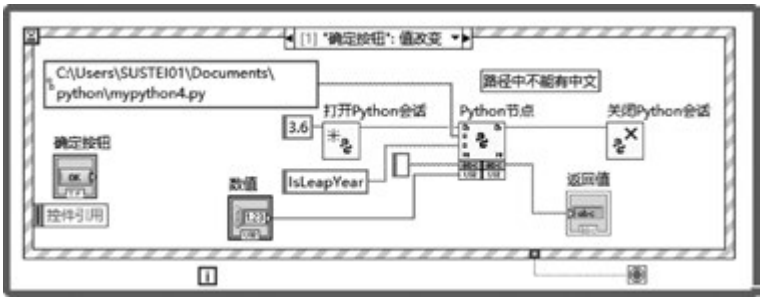


图 3-67　LabVIEW 中调用 Python 函数程序框图

运行程序，输入拟判断的不同年份值，单击“确定”按钮，LabVIEW 中调用 Python 判断是否闰年的程序执行结果如图 3-68 所示。

图 3-68　LabVIEW 中调用 Python 判断是否闰年

LabVIEW 调用函数节点 Python 时，不仅需要指定模块路径和函数名称，还需要指定输入参数和返回类型，这就不可避免地涉及 Python 数据类型与 LabVIEW 数据类型的对应关系问题。Python 节点支持的数据类型包括数值、数组（包括多维数组）、字符串、簇。Python 节点将整数和字符串转换为 Python 中对应的数据类型，将数组转换为列表，并将簇转换为元组。Python 与 LabVIEW 数据类型的对应关系如表 3-1 所示。

表 3-1　Python 与 LabVIEW 数据类型的对应关系

编号	LabVIEW	Python
1	数值	Number（数字）
2	字符串	String（字符串）
3	数组	List（列表）
4	簇	Tuple（元组）

函数节点 Python 仅支持数值、数组（包括多维数组）、字符串、簇这 4 种数据类型，意味着 LabVIEW 中的其他数据类型，如布尔、波形、变体等以及包含这些类型的数组和簇，都无法作为函数节点 Python 的输入参数或返回类型。因此，Python 程序设计时应避免使用和 LabVIEW 不匹配的参数类型。

3.5　设计模式

本节主要介绍项目开发实践中形成的轮询、事件响应、状态机、主从式、生产者/消费者等多种成熟可靠程序框架（设计模式）的基本原理、基本组成及其在程序设计中的典型应用。

3.5.1　轮询设计模式

1. 基本原理

轮询模式是 LabVIEW 早期常用的设计模式之一。在很多应用场景中，程序需要周期性地监测、判断相关控件取值、部分代码或子程序执行结果的变化，并根据变化进行相应的处理。轮询模式针对这一类需求，提供了一种成熟、可靠的程序设计框架。

轮询模式普遍用于监测、控制类应用程序设计，至今仍然是单片机软件开发的主流模式之一。但是轮询模式下的程序需要在每次循环中对所检测的对象进行访问和计算，极端情况下，可能仅仅为了等待某一个值的变化而执行成千上万次循环，而且在循环的过程中由于延时的引入，有时也会无法捕捉到某些检测对象的瞬时变化，导致程序执行效率比较低而且可靠性不高，所以一般仅仅适合状态缓变对象的简单监控程序设计与开发。

2. 基本组成

轮询模式基本结构一般由 While 循环结构、移位寄存器、循环延时、错误处理、停止条件等 5 部分组成。

（1）While 循环结构。用来实现连续动作或者功能的执行。

（2）移位寄存器。用来捕捉、传递每次循环过程中的错误信息。

（3）循环延时。为了避免 While 循环独占系统资源，降低系统的响应速度，While 循环中添加延时函数，为系统处理其他用户请求留出时间。

（4）错误处理。每次循环中检测是否有错误发生，包括软件运行错误、硬件设备错误等各种类型不安全因素导致的程序异常。当出现错误时，程序能够报告错误并且自动退出。

（5）停止条件。轮询模式一般采取组合逻辑判断决定是否退出程序。一是借助“停止”按钮的动作判断，二是对于循环执行过程中传递的错误信息判断。当检测到错误状态或者停止按钮操作，循环停止执行。

轮询模式的程序框图基本结构如图 3-69 所示。

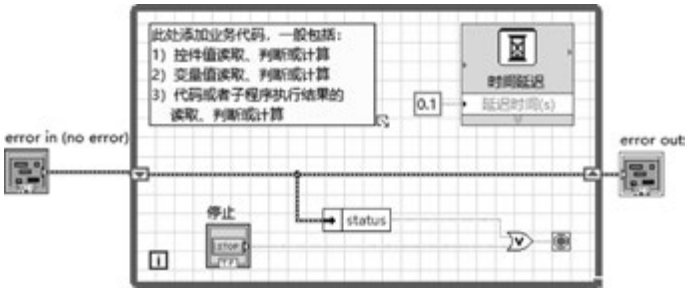


图 3-69　轮询模式的程序框图基本结构


微课视频

3. 设计实例

1）设计目标

设计开发一个数据采集系统，具备以下功能。

（1）数据采集功能。以指定范围随机数产生的方式模拟温度数据采集。

（2）开关控制功能。具有数据采集启动开关，用户开启开关，启动数据采集，关闭开关，停止数据采集。

（3）数据显示功能。能够显示采集数据的波形图，能够实时显示采集的数据值。

（4）超限报警功能。当采集数据大于设定的阈值时，蜂鸣器报警，指示灯亮，显示超限数据；非报警模式下，超限数据默认显示 0。

2）设计思路

程序实现采用轮询模式进行设计。程序前面板提供开关控件，作为启停数据采集的依据；提供布尔控件作为异常情况警示灯，红色为报警状态，默认为绿色状态；提供波形图表控件实时显示采集数据的波形；提供数值显示控件，分别显示实时采集的数据、报警时的数据取值；提供停止按钮作为结束程序的判断依据。在 While 循环中，程序判断以下 3 类状态。

（1）开关状态。如果开关打开，则以产生随机数的方式模拟数据采集工作（简化程序设计，此处重点在于轮询模式的应用），并实时显示每次数据采集所获取的数据。如果开关关闭，则停止随机数产生，模拟数据采集工作暂停。

（2）采集数据状态。当采集到数据后，程序借助波形图表显示采集数据的波形，并判断采集数是否大于设定阈值，如果大于则显示报警时采集数据的取值，警示灯亮并启动蜂鸣器

报警，否则报警数据显示 0。

（3）停止按钮单击状态。当用户单击停止按钮时，程序退出。

3）程序实现

按照程序设计功能要求以及问题解决思路，设计轮询模式下的数据采集仿真程序前面板如图 3-70 所示。

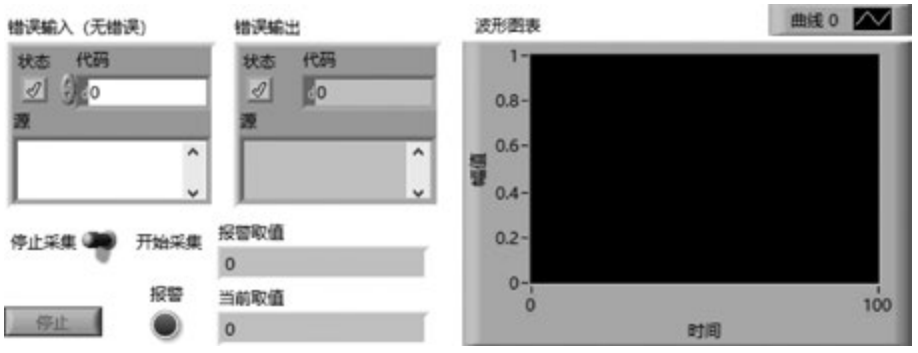


图 3-70　轮询模式下的数据采集仿真程序前面板

程序框图中，While 循环内以 1ms 的时间间隔实现对于开关“采集数据”的状态检测。当开关打开时，以产生随机数的方式模拟数据采集，在显示采集数据的基础上，进一步判断采集数据是否大于或等于 0.8，如果超出指定阈值，则蜂鸣器报警。程序中同时检测错误信息和“停止”按钮状态，作为结束程序运行的依据。轮询程序中开关“采集数据”打开时的程序框图如图 3-71 所示。

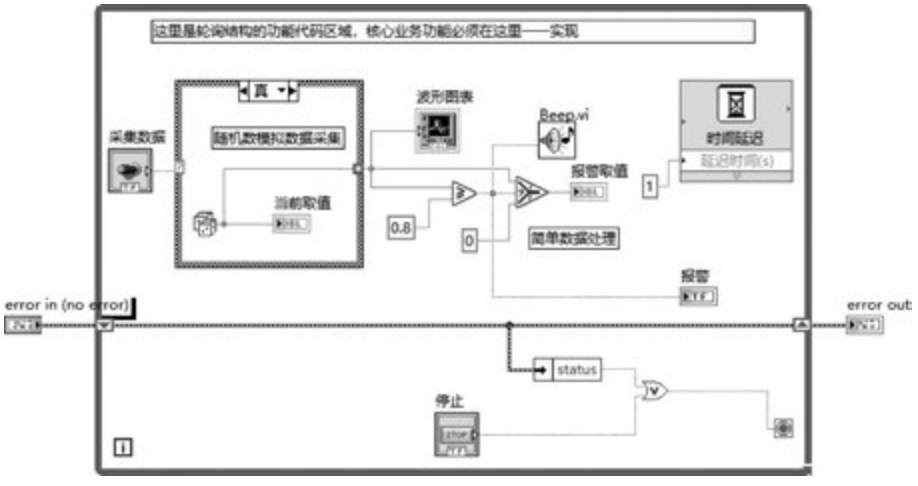


图 3-71　轮询程序中开关“采集数据”打开时的程序框图

当开关关闭时，轮询程序中开关“采集数据”关闭时的程序框图如图 3-72 所示。

单击工具栏中的“运行”按钮 ，测试程序功能。

程序启动后，操纵水平摇杆开关“开始采集”，拨动其至右侧启动数据采集（产生随机数），波形图表、数值显示控件“当前取值”实时显示采集的数据。当采集数据大于设定的阈值时，蜂鸣器报警，LED 指示灯亮，数值显示控件“报警取值”显示报警时采集的数据值。当

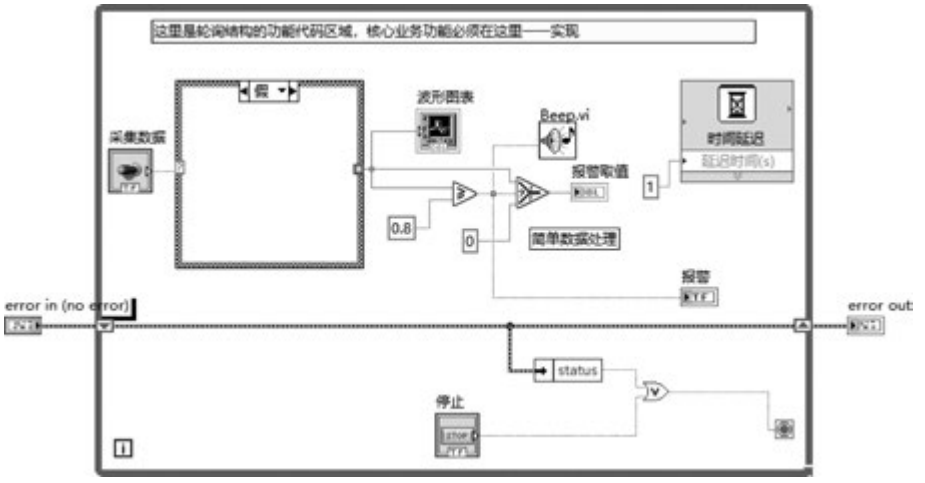


图 3-72　轮询程序中开关“采集数据”关闭时的程序框图

采集数据小于设定的阈值时，LED 恢复至默认状态，数值显示控件“报警取值”显示 0。

轮询程序执行结果如图 3-73 所示。

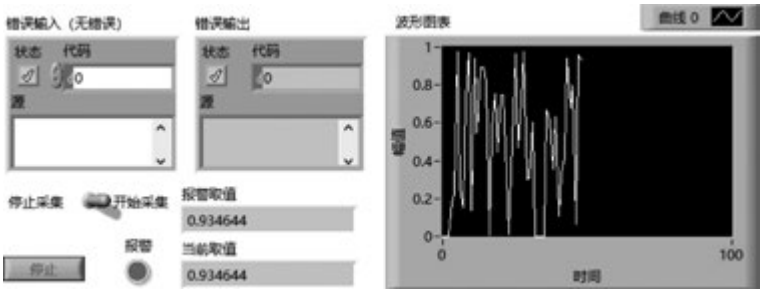


图 3-73　轮询程序执行结果

用户单击“停止”按钮后，程序正常结束运行。这说明轮询模式的基本结构是可靠的，可以在应用程序开发中借鉴使用。

3.5.2　事件响应设计模式

1. 基本原理

事件响应模式是 LabVIEW 中极其重要且极为经典一种程序结构。相比于轮询模式反反复复查询某种状态、执行效率极为低下的现象，事件响应模式则类似于“中断”，只有当事件发生之后，CPU 才对事件进行处理，即为响应事件，执行相关操作，实现程序相关功能。事件未发生时，CPU 处理其他进程相关事务。这种结构 CPU 使用效率高达 100%。

事件响应模式将程序功能分解在几个不同的事件处理子框图中，提供了有限屏幕区域中更多代码编写与功能实现的可能，使得程序逻辑更加清晰、可读性更强。事件响应模式适合程序中人机交互操作比较频繁的应用程序设计。但是这种设计模式忽略了事件之间的逻辑关系，比如事件之间的“互锁”或者“时序”特征，程序编写中必须对于基本事件响应模式进行必要的改良，以避免异常发生。

2. 基本组成

事件响应设计模式一般至少由 While 循环、事件结构两部分组成。

(1) While 循环。用来持续执行事件的监测和对应的处理功能。

(2) 事件结构。内嵌于 While 循环之中，用来监测所有的用户事件，提供程序需要处理的各类事件分支处理框架。

事件响应模式的程序框图如图 3-74 所示。

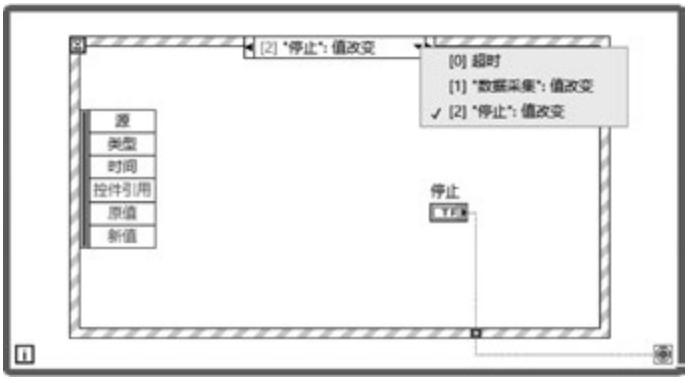


图 3-74　事件响应模式的程序框图

图 3-74 中所示的事件响应程序基本结构中，程序需要循环监测并处理以下 3 个事件。

(1) 超时事件。程序在指定的时间内没有发生任何人机交互动作而触发。

(2) "数据采集"：值改变事件。按钮控件"数据采集"取值发生变化而触发。

(3) "停止"：值改变事件。按钮控件"停止"取值发生变化而触发。

若无这 3 个事件发生，程序不做任何其他处理。3 个事件的任何一个事件发生，程序则进入对应的事件处理子框图，执行相应的业务代码，实现对应的业务功能。

3. 设计实例


微课视频

1) 设计目标

设计开发一个数据采集系统，具备以下功能。

(1) 采集控制。具有数据采集、停止运行的操作按钮，用以启动数据采集、退出应用程序。

(2) 数据采集。监听程序界面数据采集按钮的状态，当按钮按下时，以指定范围随机数产生的方式模拟温度数据采集，每次连续采集 100 个数据。

(3) 异常检测。程序能够根据操作界面输入的高温阈值、低温阈值判断每次采集的数据是否存在异常。

(4) 数据显示。能够显示采集数据的实时曲线，能够实时显示采集的数据值。

(5) 文件存储。能够将采集温度数据的序号、数据值、异常情况等信息写入电子表格文件永久保存。

2) 设计思路

程序前面板提供数值输入控件，用以设置高温阈值、低温阈值；提供数值显示控件用以显示当前采集数据；提供圆形指示灯用以指示是否发生高温警报、低温警报；提供字符串显示控件用以显示报警信息；提供波形图表用以显示采集数据的波形；提供按钮控件用以

启动数据采集、结束应用程序。

程序框图采用基于事件响应模式进行程序设计。

为了进一步贴近工程化应用，对基本事件响应结构进行优化处理，即借助顺序结构将程序分为“初始化”“主程序”两部分的功能。

（1）初始化部分。顺序结构第一帧，模拟硬件初始化，设置电子表文件的操作路径、程序界面显示信息的初始化。

（2）主程序部分。顺序结构第二帧，应用程序的主功能实现体现在这一帧，以事件响应模式处理用户界面的数据采集、停止程序等按钮操作事件，完成程序全部功能。

主程序中，当监测到“数据采集”按钮控件的单击事件时，程序以产生指定范围的随机整数方式模拟数据采集工作。程序连续采集 100 个数据，并根据操作界面用户输入的温度上下限阈值判断数据是否异常，如果高于上限阈值，显示高温警报且高温警报指示灯亮，如果低于下限，显示低温警报且低温警报指示灯亮，否则显示正常。采集完毕，所有采集的数据及其序号、异常情况 3 种信息形成电子表格文件的记录。当监测到“停止程序”按钮控件的单击事件时，程序结束运行。

3）程序实现

按照程序设计功能要求以及问题解决思路，设计事件响应模式的数据采集程序前面板，如图 3-75 所示。

程序框图实现时，初始化帧完成前面板显示控件的初始赋值，清空波形图表显示内容，生成采集数据存储的文件路径和文件名称，事件响应模式的数据采集程序初始化帧如图 3-76 所示。

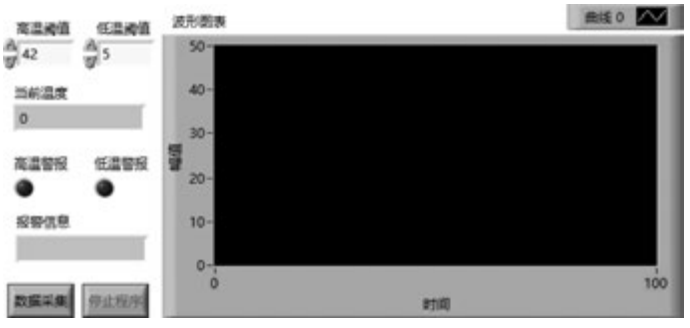


图 3-75　事件响应模式的数据采集程序前面板

图 3-76　事件响应模式的数据采集程序初始化帧

程序初始化帧后的第 2 帧为主程序帧，由 While 循环结构内嵌事件结构组成，事件结构处理按钮控件“数据采集”事件及按钮控件“停止”事件。

其中“数据采集”按钮值改变事件处理程序子框图如图 3-77 所示。

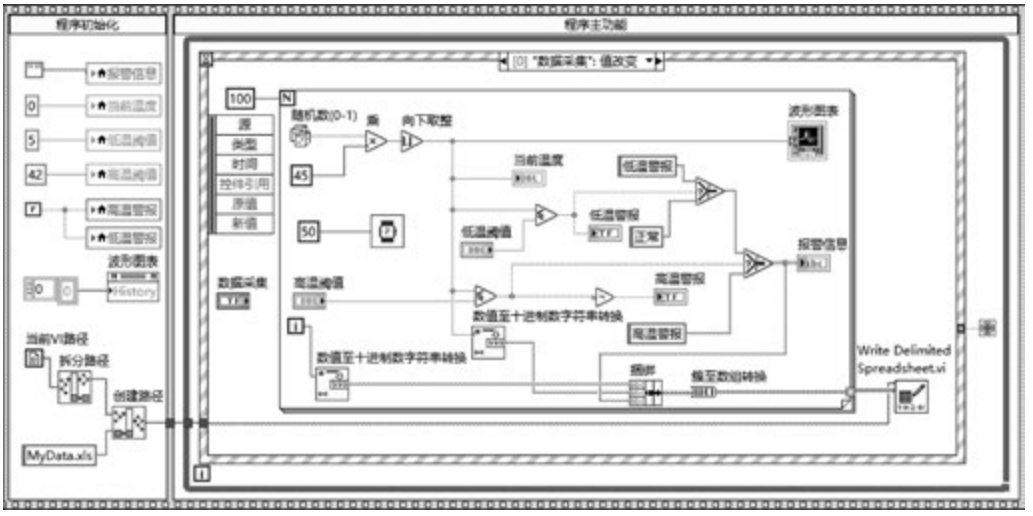


图 3-77　“数据采集”按钮值改变事件处理程序子框图

“停止”按钮值改变事件处理程序子框图如图 3-78 所示。

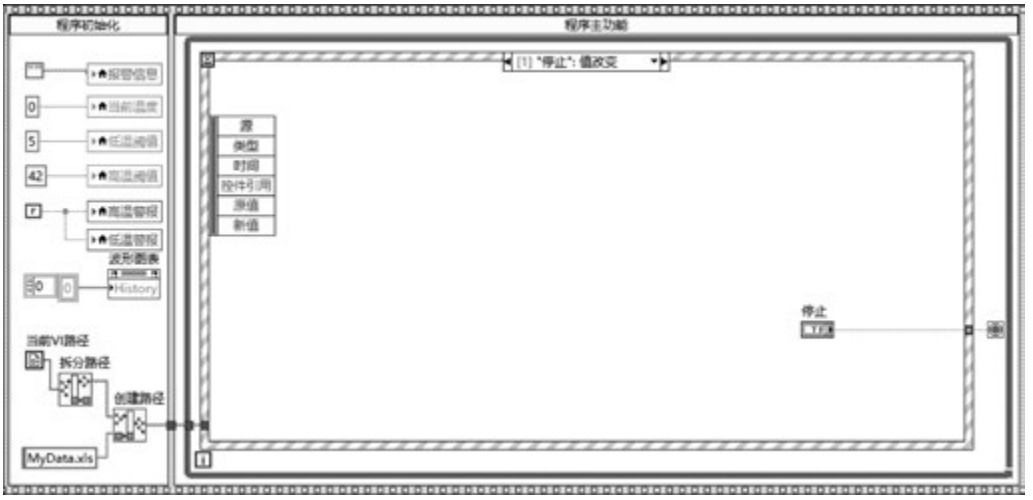


图 3-78　“停止”按钮值改变事件处理程序子框图

单击工具栏中的“运行”按钮 ，测试程序功能。

程序运行时默认高温阈值为 42，低温阈值为 5，用户可以在进一步操作之前修改阈值数据。初始状态，当前温度显示 0，高温警报和低温警报指示灯灭，报警信息显示为空，波形图表显示内容为空，事件响应模式的数据采集程序运行初始界面如图 3-79 所示。

单击“数据采集”按钮，采集数据的结果如图 3-80 所示。

单击“停止程序”按钮，在数据采集任务完成的情况下，程序可以正常地结束运行。此时打开 VI 程序所在的文件夹，可以发现程序生成的电子表格文件 MyData. xls。

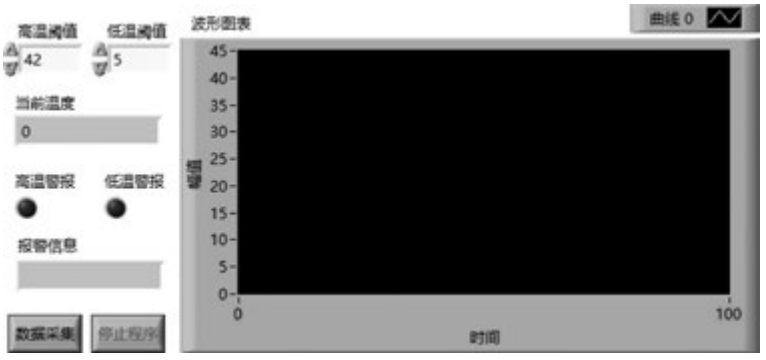


图 3-79　事件响应模式的数据采集程序运行初始界面

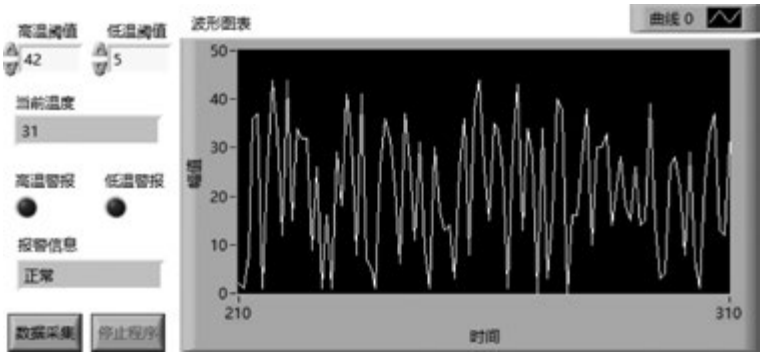
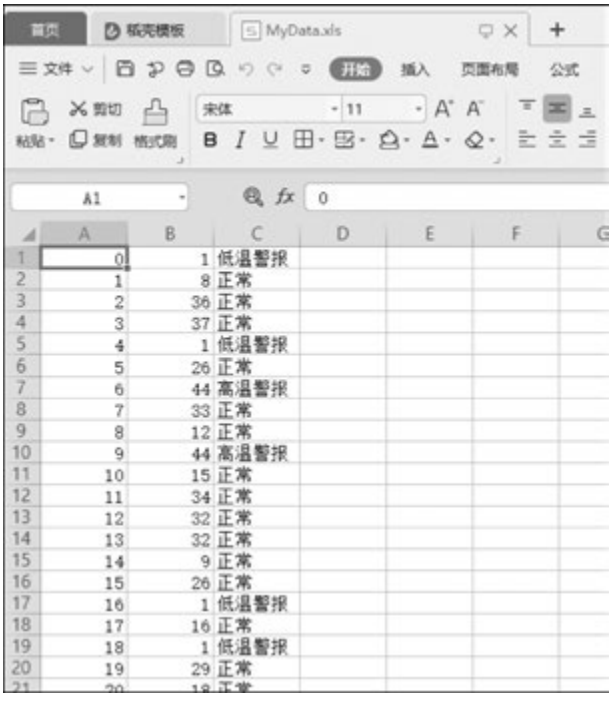


图 3-80　采集数据的结果

打开电子表格文件，其存储的数据内容如图 3-81 所示。



	A	B	C
1	0	1	低温警报
2	1	8	正常
3	2	36	正常
4	3	37	正常
5	4	1	低温警报
6	5	26	正常
7	6	44	高温警报
8	7	33	正常
9	8	12	正常
10	9	44	高温警报
11	10	15	正常
12	11	34	正常
13	12	32	正常
14	13	32	正常
15	14	9	正常
16	15	26	正常
17	16	1	低温警报
18	17	16	正常
19	18	1	低温警报
20	19	29	正常

图 3-81　电子表格文件存储的数据内容

3.5.3　状态机设计模式

1. 基本原理

状态机设计模式在其发展过程中先后演化出顺序状态机、改进状态机、标准状态机、事件状态机、超时状态机等多种形式，其中标准状态机设计模式应用最为广泛。

基于状态机设计模式的程序设计，就是将应用程序划分为有限个运行状态，程序运行过程中可根据需要在不同的状态之间任意切换和反复执行，从而实现比较复杂的程序逻辑功能。一般情况下，当系统执行的过程中需要以某种形式互锁进行控制或者需要按照时间顺序进行有序控制时，就可以采用状态机设计模式。状态机设计模式提供了一种方便、快捷、灵活的程序框架，是最有效的程序设计方法之一。

状态机设计模式应用的关键在于理解“起始、现态、条件、动作、次态（目标状态）、终止”表征程序状态相关的六种元素。

（1）起始。表示状态机开始运行。

（2）现态。即当前程序所处的状态。

（3）条件。又称为“事件”，当一个条件被满足时，将会触发一个动作，或者执行一次状态的迁移。

（4）动作。即条件满足后执行的动作。动作执行完毕后，可以迁移到新的状态，也可以仍旧保持原状态。动作不是必需的，当条件满足后，也可以不执行任何动作，直接迁移到新状态。

（5）次态。即条件满足后要转移的新状态。“次态”是相对于“现态”而言的，“次态”一旦被激活，就转变成新的“现态”了。

（6）终止。表示状态机结束运行。

清晰明了的状态图，是设计代码逻辑架构的前提。一般先绘制完成状态图，然后再使用编程语言去实现。绘制状态图的一般步骤是：分析系统业务功能，寻找主要的状态，确定状态之间的转换，细化状态内的活动与转换条件。

状态图的表示分为表格式表示法和图形化表示法 2 种，比较常用的是图形化表示法。图形化表示法借助如图 3-82 所示的 4 种主要图元表示状态图。

图 3-82　表示状态图的 4 种主要图元

4 种主要图元即可表示应用程序中存在的主要状态以及状态之间转移和转移条件。图 3-83 给出了一个 n 种状态转移的状态图示例。

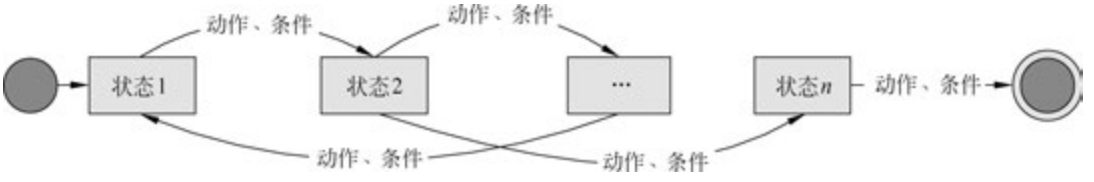


图 3-83　状态图示例

2. 基本组成

标准状态机设计模式基本结构一般由 While 循环结构、移位寄存器、枚举常量、条件结构、选择函数等几部分组成。

(1) While 循环结构。用来实现连续动作或者功能的执行。

(2) 移位寄存器。用来传递程序下一个需要执行的状态，While 循环左移位寄存器为程序执行的“现态”，右移位寄存器为程序执行的“次态”。

(3) 枚举常量。该常量包含了系统所有可能的状态，每次状态转移都会选择其中一个指定的状态。

(4) 条件结构。条件结构的每个分支对应程序一种可能的运行状态。在条件结构的每个分支(程序的每一个运行状态)中，不仅需要编写对应的功能代码，还要确定该状态下程序满足何种条件转移至哪种状态。

(5) 函数节点“选择”。这是一种比较运算节点，是典型的双分支执行路径判断，用来确定标准状态机从“现态”转移至哪个“次态”。当满足某种条件时，程序转移至一种状态，否则程序转移至另一种状态(这里状态用枚举常量的不同取值表示)。

标准状态机设计模式的程序框图基本结构如图 3-84 所示。

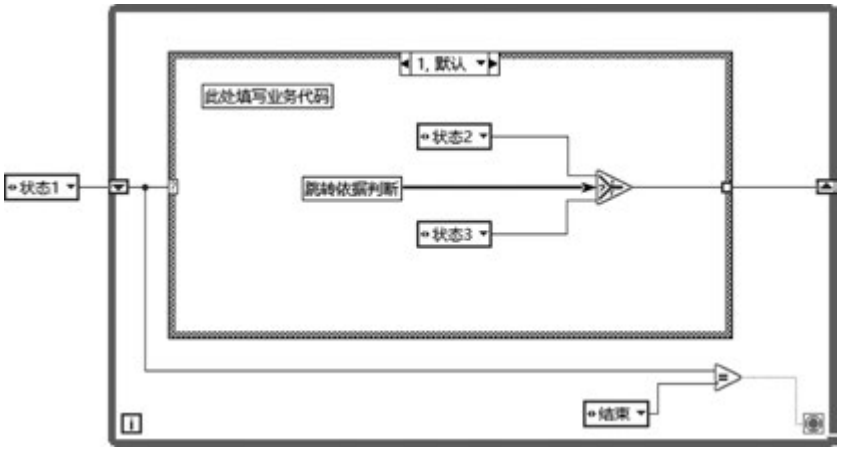


图 3-84　标准状态机设计模式的程序框图基本结构

标准状态机使用循环结构的移位寄存器来选择需要执行的下一个状态，不需要与用户之间进行交互动作，多用于一些具有显著时序特征的多任务应用程序设计。总的来说，标准状态机具有结构简单、设计方便、无须人工干预介入、可以自行决定程序执行路径、可以在有限的屏幕区域内实现丰富的功能等显著优点，但是由于标准状态机设计模式要求前状态只能由上一个状态来决定，因此当系统需要人机交互或者其他复杂操作时，这种设计模式则会无能为力，必须对标准状态机设计模式进行改造或者采取其他设计模式。

3. 设计实例


微课视频

1) 设计目标

设计开发一个数据采集系统，具备以下功能。

(1) 数据采集。以指定范围随机数产生的方式模拟温度数据采集；能够实时显示采集数据的波形图，能够实时显示采集的数据值。

（2）采集数据分析。当采集数据大于设定的阈值时，指示灯亮、显示超限数据；非报警模式下，超限数据默认显示 0。

（3）异常情况处置。当采集数据出现异常时，保存相关数据信息。

（4）停止程序。根据指令停止持续执行。

（5）循环处理。程序运行后自动在数据采集、数据分析、异常处置等状态之间转移，无须人工介入，直至检测到结束程序运行命令。

2）设计思路

基于标准状态机进行程序设计。根据需求，将应用程序划分为以下 4 个状态。

（1）数据采集状态。以指定区间随机数产生的方式模拟数据采集，获取数据并完成数值显示、图表显示后，结束数据采集状态，程序自动进入数据分析状态。

（2）数据分析状态。根据指定的阈值（上限、下限）判断采集数据是否越限。如有越限情况，根据越限情况生成报警信息并显示，程序转入数据记录状态；如无越限情况，程序转入延时等待状态。

（3）数据记录状态。取系统当前时间、采集数据值以及报警信息等数据，合并生成一条数据记录字符串，写入指定的文本文件中，完成任务后程序转入延时等待状态。

（4）延时等待状态。如果本次业务流程已用时间计时器到达指定的时间间隔，且用户未单击停止程序执行，则程序转入数据采集状态，开始新一轮业务流程；否则继续停留在本状态；当用户停止程序执行时，退出应用程序。

根据上述数据采集业务的状态划分以及跳转方案，设计如图 3-85 所示的数据采集程序状态转移图。

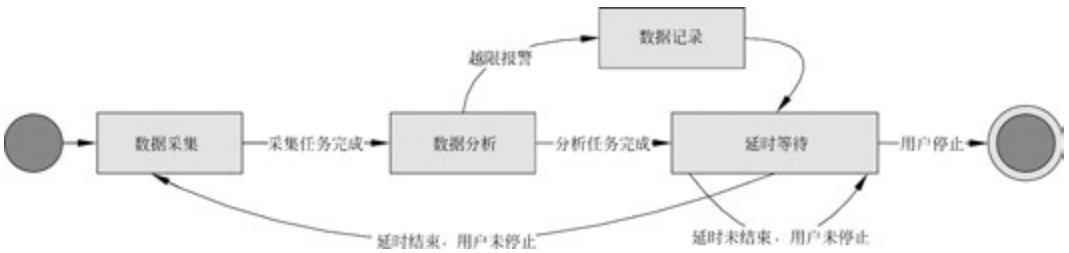


图 3-85　数据采集程序状态转移图

在数据采集程序状态转移图中，程序初始化结束后直接进入“数据采集”状态，该状态任务完成后无条件转移至“数据分析”状态。数据分析状态判断采集数据是否越限，如果越限则转移至“数据记录”状态，否则转移至“延时等待”状态。“数据记录”状态任务完成后无条件转移至“延时等待”状态。“延时等待”状态中若无停止按钮单击事件，则转移至“数据采集”状态，完成一个数据采集周期，重新开始新一轮数据采集，直至用户单击“停止”按钮。

为了进一步贴近工程化应用，对基本状态机结构进行优化处理，即借助顺序结构将程序分为两部分的功能。

第一部分为顺序结构第一帧，完成应用程序的初始化。模拟硬件初始化（温度显示 0，波形图显示空，报警信息显示空字符串，计时控件重置为真），打开存储数据的文本文件，程序界面显示信息的初始化。

第二部分为顺序结构第二帧，属于应用程序的主程序部分。应用程序的主功能在这一

帧以标准状态机设计模式完成。

为了顺利实现状态转移过程中数据的传递，While 循环框架设置以下 4 组移位寄存器，实现程序从“现态”到“次态”转移过程中有关数据的更新和传递。

第一组移位寄存器为报警信息寄存器，初始值为字符串常量。

第二组移位寄存器为实时数据寄存器，初始值为数值常量 0。

第三组移位寄存器为状态转移寄存器，初始值为枚举常量的“数据采集”选项。

第四组移位寄存器为计时控件重置信号寄存器，初始值为布尔常量“真”。

3）程序实现

按照程序设计功能要求以及问题解决思路，设计基于标准状态机设计模式的数据采集程序前面板如图 3-86 所示。

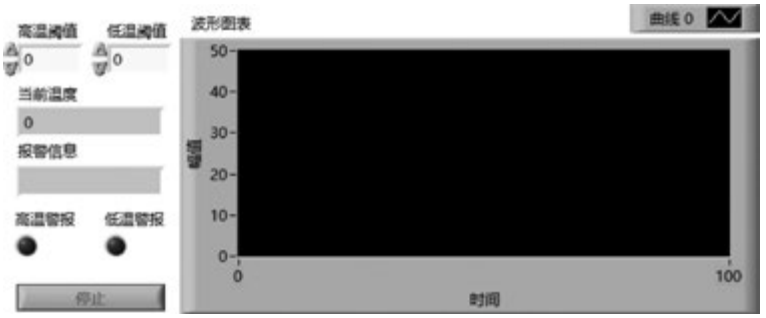


图 3-86　基于标准状态机设计模式的数据采集程序前面板

对应的顺序帧框架下的程序框图的总体结构如图 3-87 所示。

第一帧完成程序界面有关控件的初始化赋值，打开程序运行过程中需要操作的文件对象，初始化帧程序框图如图 3-88 所示。

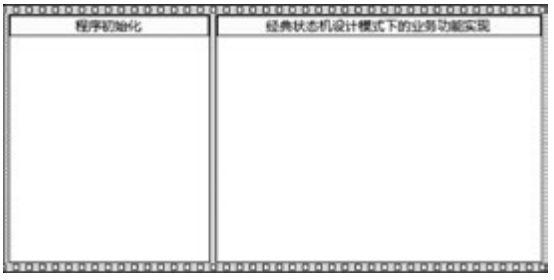


图 3-87　顺序帧框架下的程序框图的总体结构

图 3-88　初始化帧程序框图

第二帧为基于标准状态机设计模式的程序主体功能实现，程序按照前期设计的状态图在数据采集、数据处理、数据记录、延时等待 4 个状态中循环切换，其中“数据采集”状态对应的程序子框图如图 3-89 所示。

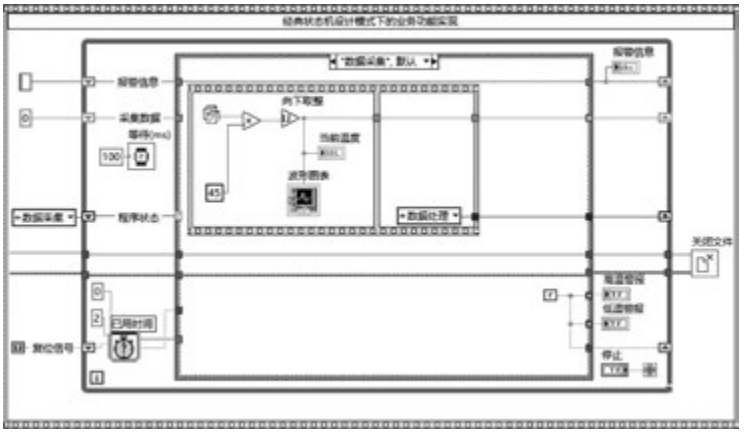


图 3-89　“数据采集”状态对应的程序子框图

第二帧中状态机在“数据采集”状态完成工作任务后，无条件转移至“数据处理”状态。在“数据处理”状态中检查当前采集数据与设定阈值关系，若未见异常则切换至“延时等待”状态，若出现异常则切换至“数据处理”状态。“数据处理”状态完整的程序子框图如图 3-90 所示。

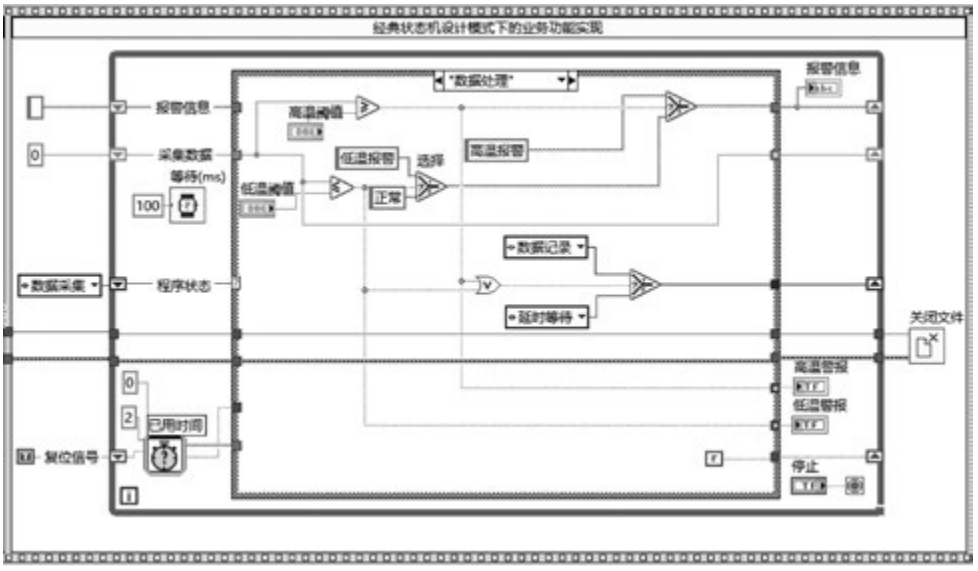


图 3-90　“数据处理”状态完整的程序子框图

程序处于“数据记录”状态时，将设定格式的报警数据写入文件，然后无条件转移至“延时等待”状态。“数据记录”状态完整的程序子框图如图 3-91 所示。

程序处于“延时等待”状态时，判断当前状态已用时间是否达到延时间隔，如果达到，则切换至“数据采集”状态，开启新一轮工作状态循环，否则继续处于“延时等待”状态。“延时等待”状态完整的程序子框图如图 3-92 所示。

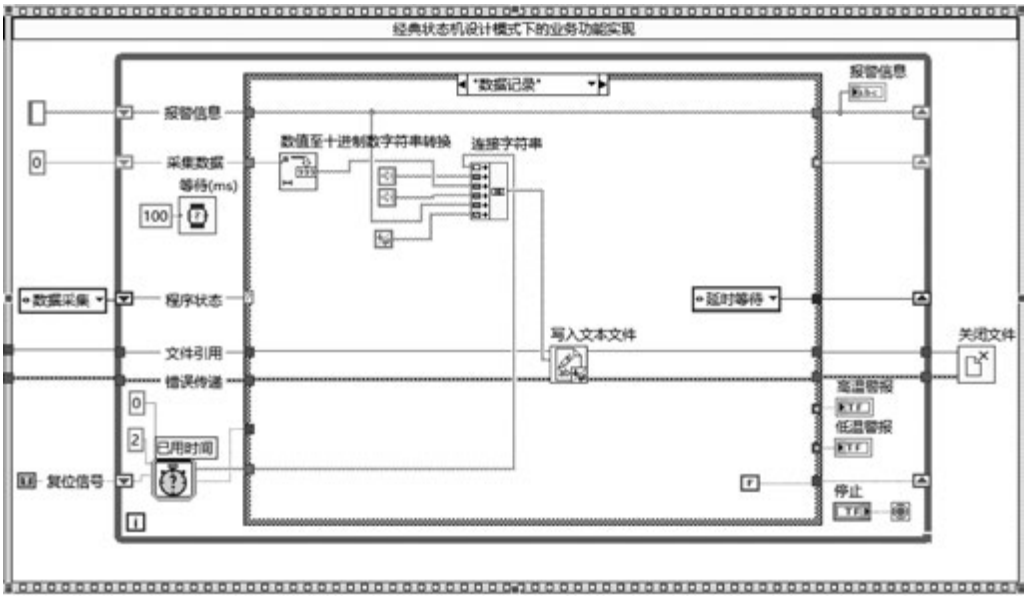


图 3-91 “数据记录”状态完整的程序子框图

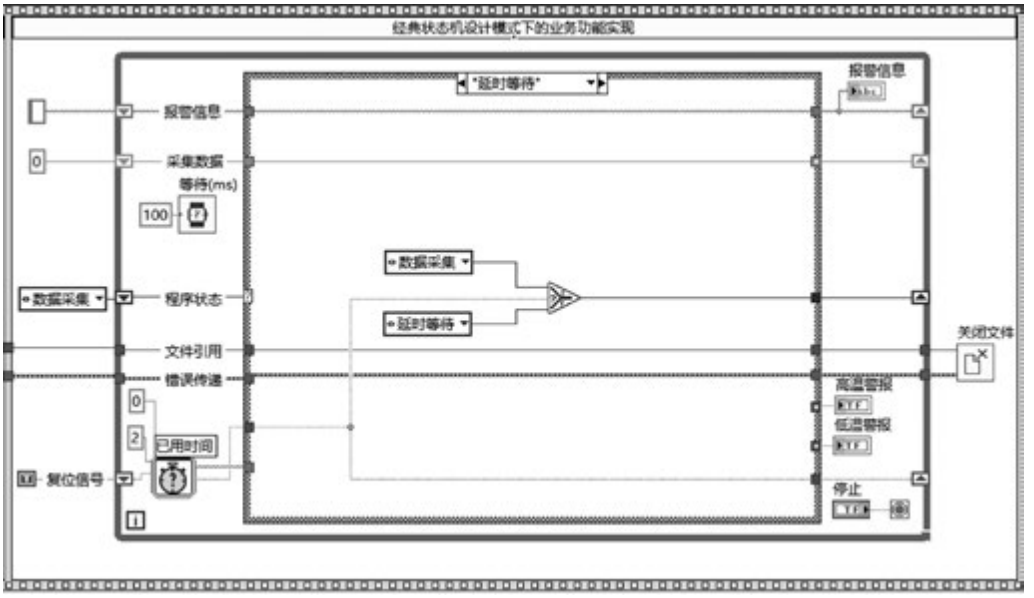


图 3-92 “延时等待”状态完整的程序子框图

单击工具栏中的“运行”按钮 。基于标准状态机设计模式的数据采集程序运行效果如图 3-93 所示。

程序运行中的曲线显示、数值显示按照指定时间间隔刷新，说明状态机工作机制运行正常。检查程序文件所在文件夹，可以发现已经自动生成数据文件 Data.txt，文件内容为历次越限数据信息，程序运行过程中以文本文件形式记录的异常数据如图 3-94 所示。

单击程序界面中的“停止”按钮，程序可以正常地结束运行，说明基于标准状态机设计模式的数据采集程序框架是可靠的，可以在应用程序开发中借鉴使用。

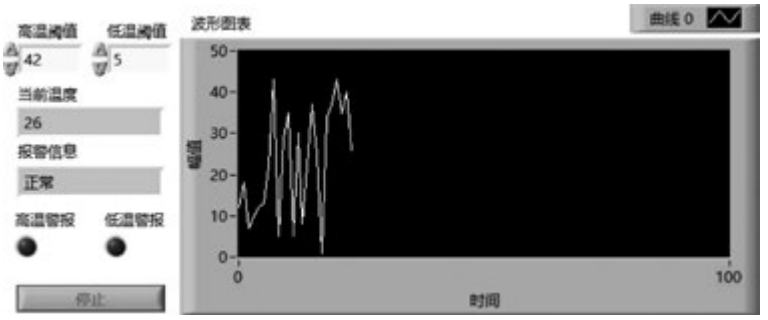


图 3-93　基于标准状态机设计模式的数据采集程序运行效果

图 3-94　程序运行过程中以文本文件形式记录的异常数据

3.5.4　主从式设计模式

1. 基本原理

主从式设计模式本质上是一种多线程程序设计模式，采取多循环的模块式结构，其每个循环代表着并行执行的一个任务，主从式设计模式提供的程序结构可以有效控制各个任务的同步执行。

主从式设计模式中的多个循环，只有一个循环称为主循环，又称主线程或者主任务，其他循环称为从线程或者从任务。在多个线程并行执行过程中，只有主线程有权利发布数据，从线程只能被动响应。主线程没有发布新的数据时，所有的从线程都是在等待数据。一旦主线程发布数据或者命令，所有从线程被唤醒，响应命令，执行相关处理任务，完成任务后又处于休眠等待的状态。主从式设计模式中主线程具有强制销毁所有从线程的能力，一般通过错误处理机制满足这一要求。

主从式设计模式借助通知器协调多线程同步工作，由于通知器的消息传递具有极强的实时性，各个从线程工作过程中，完全处于被动状态，即无消息不工作，有消息动起来，属于典型的软同步操作。

值得注意的是，通知器并不缓冲已经接收的消息，实际应用过程中应该注意各个从线程的负担不宜过重，否则在新消息到来时，如果从线程业务功能尚未处理完毕，那么从线程将出现旧消息尚未读出，新消息就已经将其覆盖，从而造成消息丢失的情况。

电子信息领域的应用系统开发，存在以“数据采集”为前提，“信号处理”“数据分析”“信息显示”“信息存取”“界面交互”等业务同步并发执行的场景。作为一种低成本高性能的多任务同步工作手段，主从式设计模式在数据采集领域具有独特优势。

2. 基本组成

主从式设计模式基本结构一般由多循环结构、通知器、程序停止策略等部分组成。

（1）多循环结构。多循环应用程序中的一个循环称为主循环或主线程，一般用于多线程的指挥和调度；其他一个或者多个循环称为从循环或从线程，从线程接收主线程的指令，完成相应的工作任务，实现预设的功能。

（2）通知器。主要用于多个线程之间的数据传递。主线程中使用通知器发送通知，从线程中使用通知器接收主线程的消息。使用通知器的最大优势在于从线程中未接到主线程的消息之前，程序挂起，不再执行，直至接收到主线程消息，因而能以比较简单的方式实现多个线程中任务的同步执行。

（3）程序停止策略。主从式设计模式编写的程序停止运行时，主线程、各个从线程均需结束运行，程序才能优雅地退出，因此采取何种策略停止应用程序的各个线程，是主从式设计模式的应用程序一项重要任务。常用的策略是组合式停止方案——主线程结束运行时向各个从线程发送停止运行命令；从线程接收到通知器停止运行命令或者引用通知器出现错误（主线程结束运行销毁从线程引用的通知器资源，导致从线程等待通知函数节点发生异常，输出错误信息），此两种情况中任何一种发生，都将结束从线程的执行。

主从式设计模式的程序框图基本结构如图 3-95 所示。

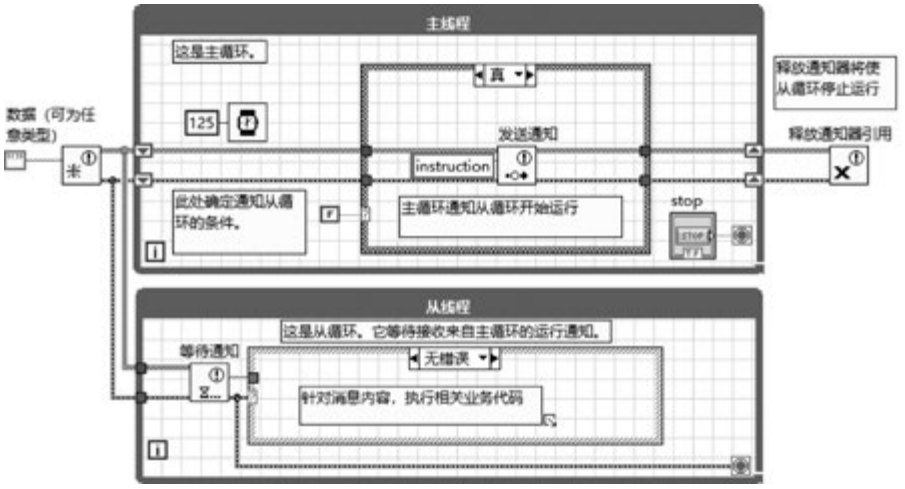


图 3-95　主从式设计模式的程序框图基本结构

在图 3-95 中所示的程序框架中，程序一开始运行，使用函数节点“获取通知器引用”，创建用于协调多线程运行的通知器，并为其分配资源，同时将通知器的引用传递给从线程。

当主线程监测程序状态符合消息发送条件时，调用函数节点“发送消息”，向从线程发送消息。

从线程调用函数节点“等待通知”在未接收到主线程消息的情况下处于挂起状态，一旦主线程发生的通知消息到来，则自动停止等待，继续执行相应的程序代码，实现从线程功能。

主线程一旦结束运行，则销毁已经创建的通知器，释放通知器在内存中所占的资源。此时从线程并未结束，从线程中会继续调用的函数节点“等待通知”，而该函数节点引用的通知器已经销毁，导致函数节点输出错误信息。这个错误信息可以作为结束从线程运行的依据。

3. 设计实例


微课视频

1）设计目标

设计开发一个数据采集系统，具备以下 4 种功能。

（1）数据采集控制。程序检测数据采集开关状态决定是否开启数据采集，检测停止按钮状态决定是否终止程序运行。

（2）采集数据显示。能够根据指定的时间间隔采集数据、显示采集数据的波形图，能够实时显示采集的数据值。

（3）异常报警功能。当采集数据大于或小于设定的阈值时，对应的指示灯亮，并且显示实时采集的数据值，显示异常数据的报警信息。

（4）数据分析处理。能够对于采集数据进行实时的时域分析，获取采集数据的平均值、标准差、最大值、最小值等信息。

在完成上述功能的基础上，采用合适的程序结构，保证数据采集、显示、报警、处理等任务的同步执行。

2）设计思路

程序设计按照主从式设计模式进行。根据程序的任务需求，将需要实现的功能分为“数据采集与程序运行控制”“异常分析与报警处理”“时域分析及其结果显示”3 个线程，其中“数据采集与程序运行控制”为主线程，另外 2 个为从线程。

为了进一步贴近工程化应用，对基本主从结构进行优化处理，即借助顺序结构程序分为两部分的功能：第一部分为顺序结构第一帧，完成应用程序的初始化操作；第二部分为顺序结构第二帧，基于主从式设计模式实现应用程序的主体功能。

3）程序实现

按照程序设计功能要求以及问题解决思路，设计基于主从式设计模式的数据采集程序前面板如图 3-96 所示。

图 3-96　基于主从式设计模式的数据采集程序前面板

对应的设计程序框图总体结构为 2 帧顺序帧。其中第一帧为初始化帧，完成各个控件的初始赋值、创建通知器引用等操作，对应的程序框图如图 3-97 所示。

第二帧为基于主从式设计模式的程序主体功能实现程序框图。首先创建 3 个线程，分别设置其子程序框图标签为“主线程 采集数据并同步异常报警和时域分析”“从线程 异常分析和报警处理”“从线程 数据的时域分析”，完成主从式设计模式对应的数据采集程序总体结构如图 3-98 所示。

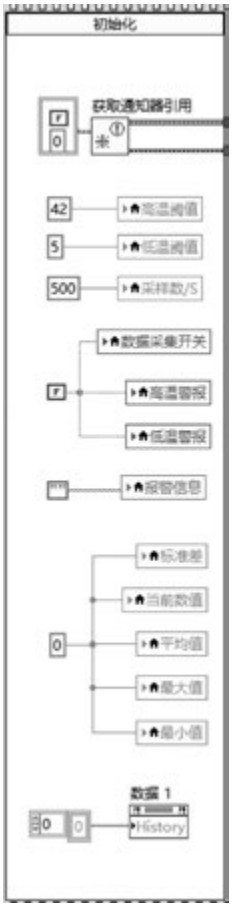


图 3-97　初始化程序框图

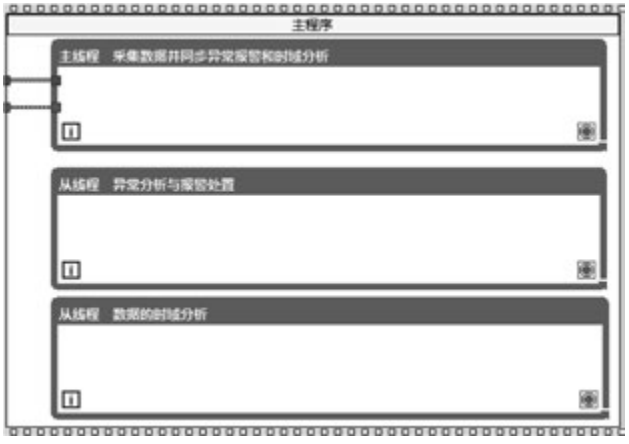


图 3-98　主从式设计模式对应的数据采集程序总体结构

第一个线程为主线程。主线程中按照指定的时间间隔检查开关“数据采集”状态，如果“数据采集”开关打开，则以簇数据封装“停止”按钮状态和采集的数据，调用函数节点“发送通知”（函数→数据通信→同步→通知器操作→发送通知）将簇数据发送出去，实现主线程向其他线程进行消息广播的功能。如果用户单击停止按钮，则退出主线程，并调用函数节点“释放通知器引用”（函数→数据通信→同步→通知器操作→释放通知器引用）销毁创建的通知器。开关“数据采集”打开时数据采集程序主线程程序框图如图 3-99 所示。

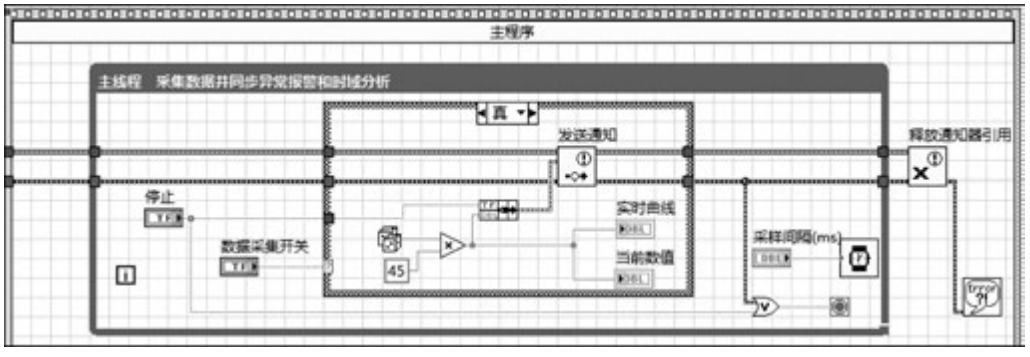


图 3-99　开关“数据采集”打开时数据采集程序主线程程序框图

当开关“数据采集”关闭时，暂停数据采集，数据采集程序主线程程序框图如图 3-100 所示。

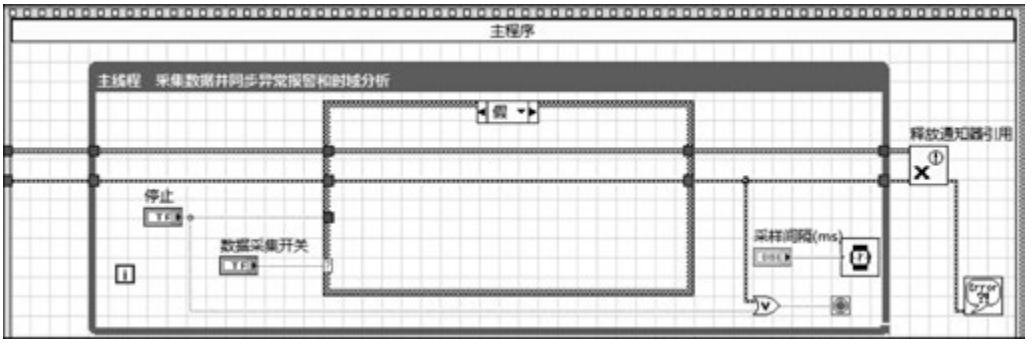


图 3-100　开关“数据采集”关闭时数据采集程序主线程程序框图

第二个线程为异常检测和报警处理线程。该线程中调用函数节点“等待通知”(函数→数据通信→同步→通知器操作→等待通知)接收来自主线程的数据。调用函数节点“解除捆绑”(函数→编程→簇、类与变体→解除捆绑)提取从线程停止命令和主线程中采集的数据，提取消息中的布尔数据成员，作为结束当前线程的依据之一；提取消息中的数值数据成员，进一步按照设定的阈值判断是否存在异常，对应的异常分析与报警处理从线程如图 3-101 所示。

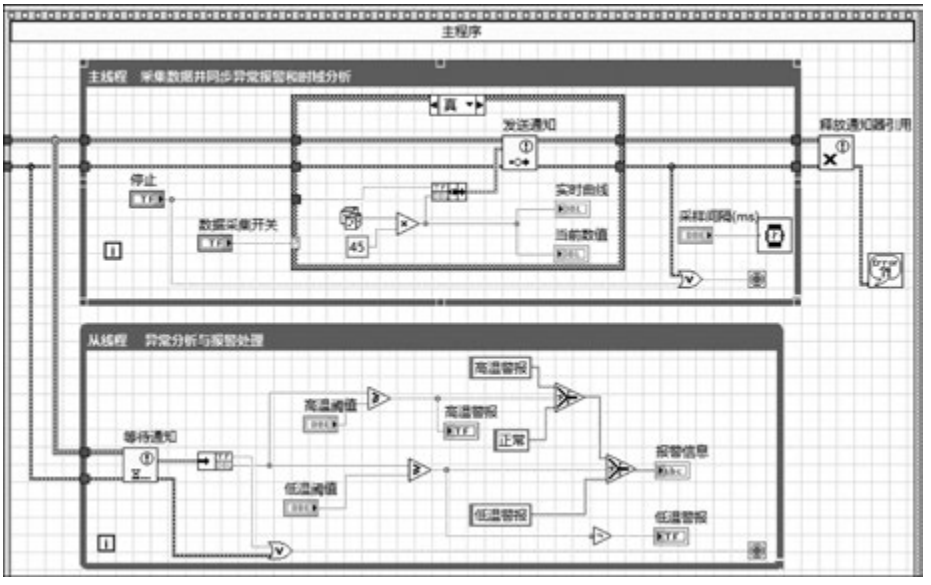


图 3-101　异常分析与报警处理从线程

第三个线程为数据时域分析线程。该线程中调用函数节点“等待通知”(函数→数据通信→同步→通知器操作→等待通知)接收来自主线程的数据。调用函数节点“解除捆绑”(函数→编程→簇、类与变体→解除捆绑)提取从线程停止命令和主线程中采集的数据，提取消息中的布尔数据成员，作为结束当前线程的依据之一；提取消息中的数值数据成员，调用函数节点“标准差(逐点)”(函数→信号处理→逐点→概率与统计→标准差)及函数节点“数组最大值与最小值(逐点)”(函数→信号处理→逐点→其他函数→数组最大值与最小值)，实现

采集数据的时域分析。对应数据时域处理从线程程序框图如图 3-102 所示。

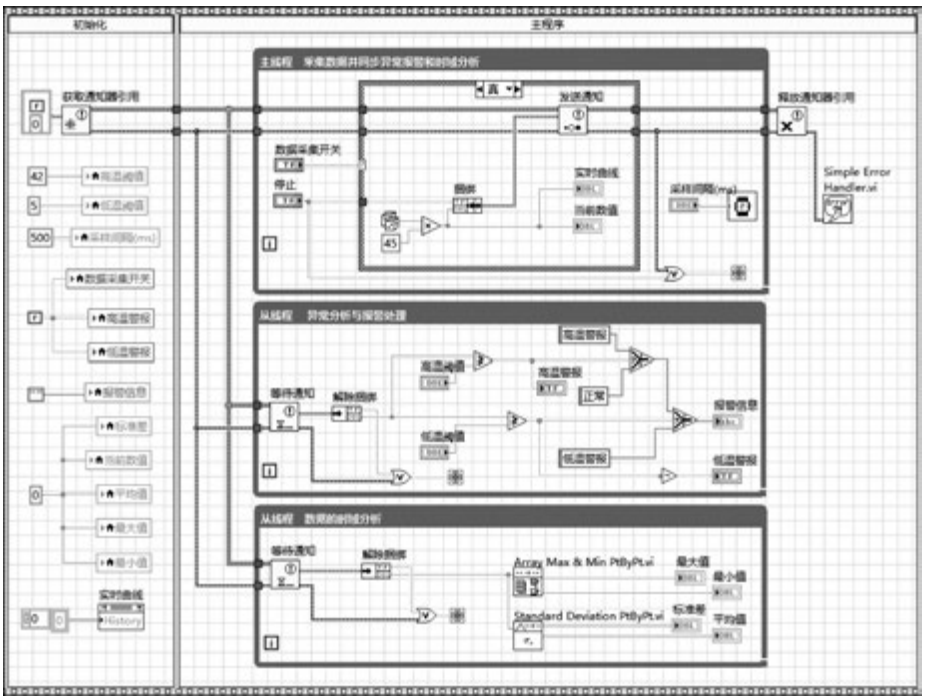


图 3-102　数据时域处理从线程程序框图

单击工具栏中的“运行”按钮 ，基于主从式设计模式的数据采集程序运行初始状态如图 3-103 所示，与程序初始化设计预期完全一致。

图 3-103　基于主从式设计模式的数据采集程序运行初始状态

打开前面板中“数据采集开关”，主线程启动温度数据的采集工作，并通过波形图表显示采集数据的实时曲线，同时主线程中通知器开始发送“停止”按钮状态和采集的温度数据所封装的簇数据。2 个从线程分别接收通知器广播的消息，解析其中的温度数据和布尔指令，分别完成异常分析与报警、时域分析功能，实现 1 主 2 从的多线程同步工作模式。基于主从式设计模式的数据采集程序运行结果如图 3-104 所示。

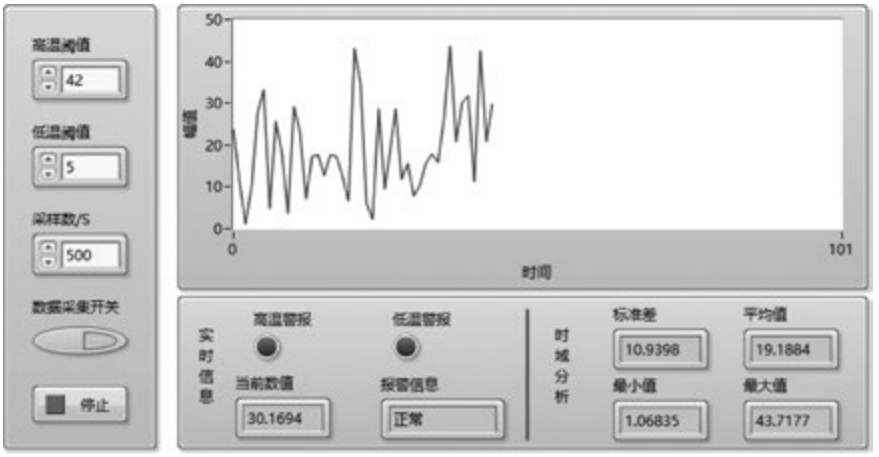


图 3-104　基于主从式设计模式的数据采集程序运行结果

程序运行过程中，单击程序界面中的“停止”按钮，主从式设计模式的应用程序可正常地结束运行，这说明主从式设计模式在多任务同步应用程序设计中是可靠的，可以在应用程序设计中借鉴使用。

3.5.5　生产者/消费者设计模式

1. 基本原理

生产者/消费者本质上是主从式设计模式的一种升级版本，它提高了不同速率线程之间的数据共享能力。与主从式设计模式类似，生产者/消费者设计模式中，主线程的任务是“生产”数据（采集数据或产生数据），一般称为生产者线程。从线程的任务是“消费”数据（处理数据），一般称为消费者线程。生产者线程和消费者线程之间依靠队列进行数据的传递。由于队列的缓冲作用，即使生产数据的速度和消费数据的速度不一致，仍然可以确保多线程并行处理过程中数据不会丢失。正因为如此，生产者/消费者设计模式广泛适用于异步并行多线程应用程序设计。

从软件的角度看，生产者线程是数据的提供方，消费者线程是数据的消费方。生产者线程和消费者线程之间存在一个数据缓冲区，其大小一般是固定的。当生产过剩而消费不足时，缓冲区的空间会随着时间的推移不断减少直至消耗殆尽。当缓冲区无剩余空间时，应该停止生产，否则将会发生异常。反之，当生产不足而消费过剩时，缓冲区的数据元素会不断减少直至为 0，此时消费者线程会处于等待状态，直至生产者线程提供新的数据，消费者线程才从缓冲区提取数据，进行进一步的处理。

2. 基本组成

生产者/消费者设计模式对应的程序结构一般由生产者/消费者线程、队列、程序停止策略等 3 部分组成。

（1）生产者/消费者线程。一般以 2 个并行执行的 While 循环方式存在。其中一个循环称之为生产者线程，另外一个循环称之为消费者线程。生产者线程产生、截获、接收、采集数据，消费者线程对数据进行分析、处理、显示、存储等操作。

（2）队列。主要用于生产者线程和消费者线程之间的数据通信/数据共享。主要工作

包括两个线程/循环执行之前的队列的创建、生产者线程中完成获取数据的入队操作。消费者线程则执行出队操作，获取队列中的数据元素，并进一步对数据进行分析、处理、显示、存储等操作。两个线程之间利用队列进行数据共享，最大的优势在于即使两个线程处理速度不一致，也不会丢失数据。

(3) 程序停止策略。程序停止运行时，生产者线程、消费者线程均需结束运行，程序才能“优雅”地退出，因此采取何种策略停止应用程序的 2 个线程，是生产者/消费者设计模式下应用程序设计的一项重要任务。常用的策略是组合式停止方案——生产者线程结束运行时向消费者线程发送停止运行命令；消费者线程接收到生产者线程发出停止命令或者引用队列出现错误(生产者线程结束运行销毁消费者线程引用的队列资源，会导致消费者线程元素出队函数节点发生异常，输出错误信息)，此两种情况中任何一种发生，都将结束从线程的执行。

生产者/消费者设计模式的程序框图基本结构如图 3-105 所示。

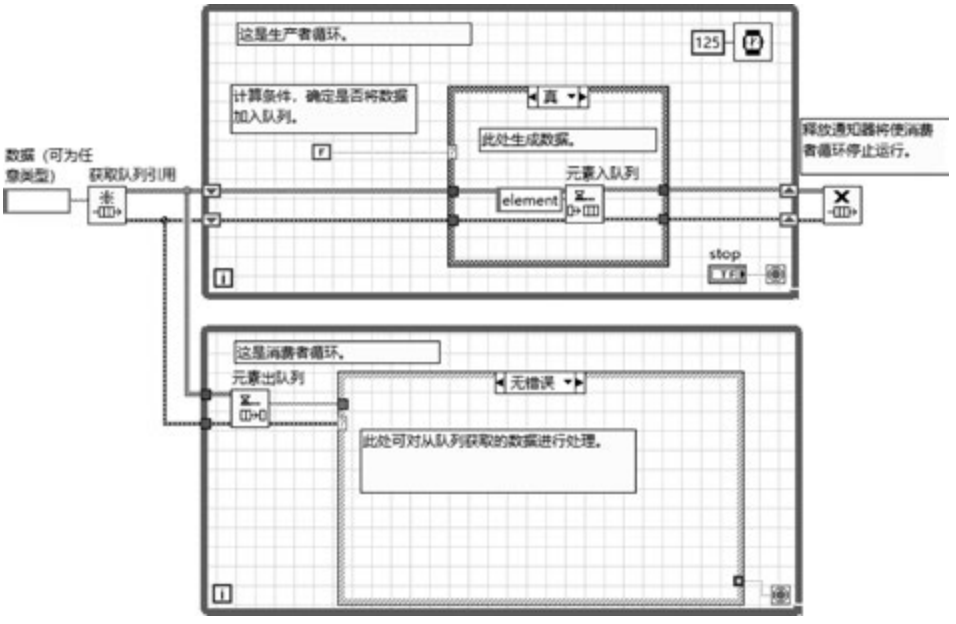


图 3-105　生产者/消费者设计模式的程序框图基本结构

图 3-105 所示的程序框图中，函数节点“获取队列引用”(函数→数据通信→队列操作→获取队列引用)创建队列，并指定了队列中数据元素的类型(当前为字符串类型，可以为任何类型)，同时将创建的队列传递给两个并行执行的循环。

当生产者线程中生成数据条件满足时，以 125ms 的时间间隔，将产生的数据(此处为 elements)进行“元素入队列”(函数→数据通信→队列操作→元素入队列)操作。

消费者线程中调用函数节点“元素出队列”(函数→数据通信→队列操作→元素出队列)，等待生产者线程入队的数据元素，如果队列不空，意味着队列中存在尚未处理的数据，则程序停止等待状态，读出队列中的一个数据元素，继续进一步的数据分析处理工作。

程序结束的方法采取了队列错误信息处理方式，单击 stop 按钮可以直接结束生产者线程，同时通过函数节点“释放队列引用”(函数→数据通信→队列操作→释放队列引用)释放创建队列时所占用的资源。而这一行为必将导致消费者线程中函数节点“元素出队”的引用

错误！可以利用这一错误信息作为退出消费者线程的依据。

3. 设计实例


微课视频

1）设计目标

设计开发一个数据采集系统，具备以下 4 种功能。

（1）数据采集控制。程序运行时检测开关状态决定数据采集任务的启停，检测“停止”按钮状态决定是否终止数据采集并退出程序。

（2）采集数据显示。能够根据指定的时间间隔采集数据、通过波形图表控件显示采集的数据波形，并能够通过数值显示控件实时显示采集的数据值。

（3）异常情况报警。当采集数据不在设定的阈值区间时，对应的报警指示灯亮，并且显示异常状态的采集数据，显示异常数据的报警信息。

（4）数据分析处理。能够对于采集数据进行实时的时域分析，获取采集数据的平均值、标准差、最大值、最小值等信息。

2）设计思路

基于生产者/消费者设计模式进行应用程序设计。针对任务需求，将程序需要实现的功能分为生产者线程和消费者线程。

为了协调生产者线程和消费者线程工作，对于基本的生产者/消费者设计模式进行改进。程序设计同时使用队列和通知器，其中队列用于生产者线程和消费者线程之间的数据共享，通知器用于生产者线程向消费者线程同步广播停止程序运行指令。

生产者线程检测程序前面板数据采集开关状态，如果打开，则以产生指定范围内随机数方式模拟数据采集工作，并将采集的数据压入队列；同时，程序按照指定的时间间隔，通过通知器不断向消费者线程广播是否结束线程运行的指令（即操作面板中“停止”按钮的状态）。

消费者线程与生产者线程引用同一个队列和通知器，调用函数节点“元素出队列”，获取采集的温度数据，并进行数据分析处理工作；同时，调用函数节点“等待通知”获取是否结束程序运行的消息。

为了实现程序“优雅”地退出目标，程序将“元素出队列”“等待通知”两种分别属于队列、通知器的函数节点的错误输出进行合并（调用函数节点“合并错误”），即无论哪个节点调用过程中出现错误，都会返回一个错误信息。然后此错误信息与通知器接收的消息合并作为消费者线程结束的条件，也就是说函数节点调用出现错误或者通知器接收到停止消息，都会结束消费线程。

为了进一步贴近工程化应用，对生产者/消费者设计模式的程序结构进行优化处理，即借助顺序结构程序分为初始化、主程序两部分的功能。

第一部分为顺序结构第一帧，完成应用程序的初始化操作。

第二部分为顺序结构第二帧，基于生产者/消费者设计模式实现应用程序的主体功能。

3）程序实现

按照程序设计功能要求以及问题解决思路，设计基于生产者/消费者设计模式的数据采集程序前面板，如图 3-106 所示。

如前所述，设计生产者/消费者设计模式的程序框图总体结构为 2 帧顺序帧。其中第一帧为初始化帧，完成各个控件的初始赋值，对应的程序框图如图 3-107 所示。

图 3-106　基于生产者/消费者设计模式的数据采集程序前面板

图 3-107　生产者/消费者设计模式的程序初始化

第二帧实现程序的主体功能。由生产者线程、消费者线程组成，借助队列实现生产者线程和消费者线程之间的数据共享，借助通知器实现 2 个线程的同步结束，对应的基于生产者/消费者设计模式的程序总体结构如图 3-108 所示。

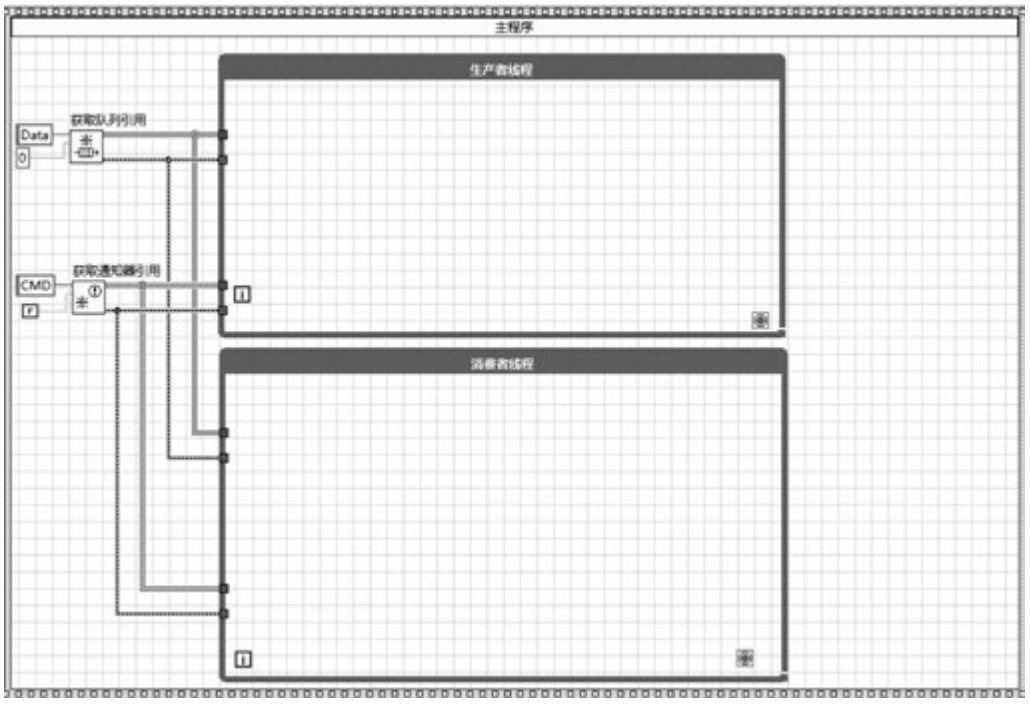


图 3-108　基于生产者/消费者设计模式的程序总体结构

生产者线程中按照指定的时间间隔，检测程序前面板中布尔类型控件“数据采集开关”状态，当开关打开时，以随机数产生的方式模拟数据采集，调用队列操作函数选板中的函数节点“元素入队列”，实现采集数据与消费者线程的共享；同时检测按钮控件“停止”的状态，调用通知器函数选板中的函数节点“发送通知”，向消费者线程广播“停止”按钮状态，作为消费者线程同步结束的判断依据之一。对应的生产者线程程序子框图如图 3-109 所示。

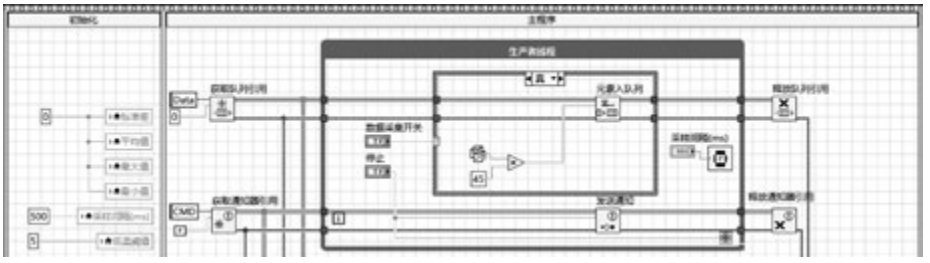


图 3-109　生产者线程程序子框图

消费者线程中，读取队列中的数据，进行数据实时显示、时域分析处理、异常检测与报警等功能；同时该线程还监测通知器消息，读取消息中的停止按钮状态值，作为消费者线程同步结束运行的依据。为了进一步增强消费者线程的健壮性，线程中读取队列节点的错误输出、通知器节点的错误输出，并合并错误信息，将错误信息和停止按钮状态的组合逻辑作为消费者线程结束的条件。对应的消费者线程程序子框图如图 3-110 所示。

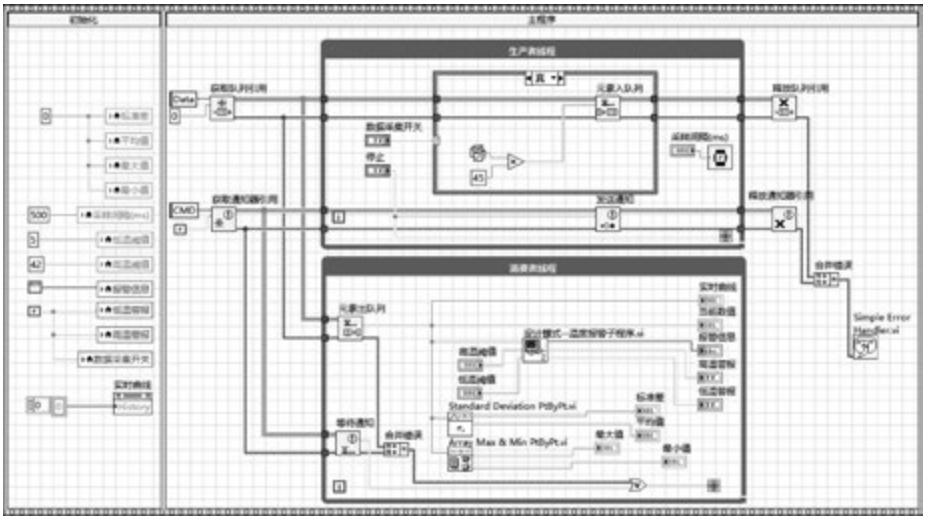


图 3-110　消费者线程程序子框图

单击工具栏中的“运行”按钮 ⇨ 。基于生产者/消费者设计模式的数据采集程序运行初始状态如图 3-111 所示，与程序初始化设计预期完全一致。

单击程序界面中“数据采集开关”开关，生产者线程启动采集温度数据任务，并对采集数据进行入队操作，借助通知器广播程序前面板中“停止”按钮状态。消费者线程通过出队操作，读取生产者线程采集的数据，并对数据进行分析处理，实现异常分析与报警、时域分析功能；同时，消费者线程实时接收生产者线程广播的“停止”按钮状态，以确定是否结束消费者线程的运行。基于生产者/消费者设计模式的数据采集程序运行结果如图 3-112 所示。

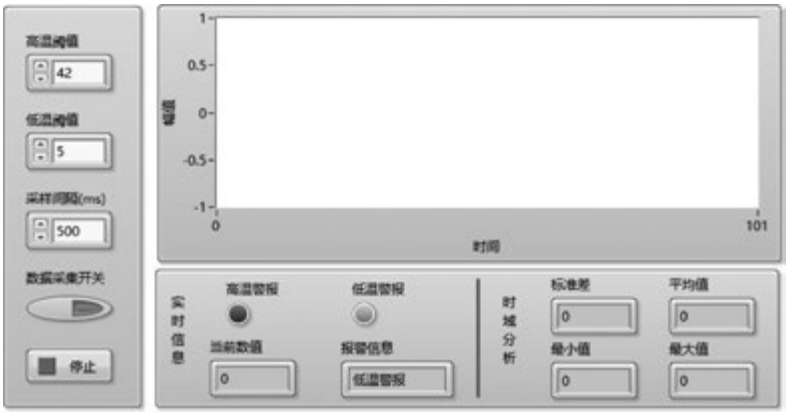


图 3-111　基于生产者/消费者设计模式的数据采集程序运行初始状态

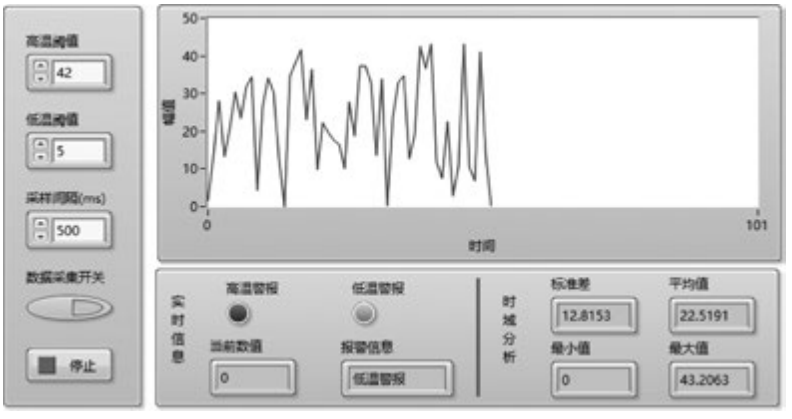


图 3-112　基于生产者/消费者设计模式的数据采集程序运行结果

无论是程序运行之初，还是开始数据采集之后，单击“停止”按钮，程序均可正常地结束运行，这说明本节案例中改进的生产者/消费者设计模式在多任务应用程序设计中是可靠的，可以在应用程序设计中借鉴使用。

思考与拓展

1. 如欲在程序执行过程中显示多路采集数据的实时曲线，使用波形图显示控件还是使用波形图表显示控件比较合适？

2. 什么是子 VI，子 VI 设计的主要步骤是什么？

3. 使用循环结构调用随机数函数模拟多路实时数据采集，并将采集数据分别存储于 CSV 格式的电子表格文件、文本文件（文本文件的一行为多路数据一次采样的结果，数据之间间隔符号可以自行定义）。

4. 分别基于 MATLAB、Python 联合编程的方式完成基于 LabVIEW 的一元二次方程 $ax^2+bx+c=0$ 根的求解。

5. 总结五类常用的 LabVIEW 设计模式各自的技术优势和适用的场景。