浙江大学

本科实验报告

课程名称：	通信原理实验
姓　　名：	黄嘉欣
学　　院：	信息与电子工程学院
系：	信息与电子工程学系
专　　业：	信息工程
学　　号：	3190102060
指导教师：	龚淑君　金向东

2022 年 4 月 20 日

专业：信息工程
姓名：黄嘉欣
学号：3190102060
日期：2022 年 4 月 20 日
地点：东四-319

浙江大学实验报告

课程名称：通信原理实验　　指导老师：龚淑君 金向东　　成绩：

实验名称：熟悉 LabVIEW　　实验类型：设计性实验　　同组学生：张维豆

装 订 线

一、实验目的

① 熟悉 LabVIEW 通信设计套件；

② 会使用 While/For 循环处理数据；

③ 熟悉多种数据类型；

④ 会使用数组，访问和处理元素。

二、实验设备

① 安装 LabVIEW 环境的电脑　　1 台；

三、实验概要

本实验课程的目的是在通用软件无线电外设（USRP）上应用、实践关于软件无线电的数字通信理论。USRP 结合了 LabVIEW 软件，该软件是一个使用模块和连线的图形编程软件。因此有必要熟悉 LabVIEW 语言并能写一些代码。

四、实验内容与步骤

① 开始：

(1) 运行电脑中的软件：NI LabVIEW NXG 2.1；

(2) 新建一个项目：点击文件/新建/项目；

(3) 在项目中添加一个新的 VI：点击文件/新建/VI；

(4) 将项目重命名为 Lab1：在窗口左上角右击未命名项目并点击重命名，窗口如图 4.1 所示；

(5) 每一个 VI 有两个窗口：前面板和程序框图。前面板是用户和 VI 进行交互的地方，它包括很多控制输入和显示输出参数端口。程序框图窗口包含从控制输入端接收

输入数据，通过连接模块进行处理，并且通过显示输出进行显示的代码（流程图）。

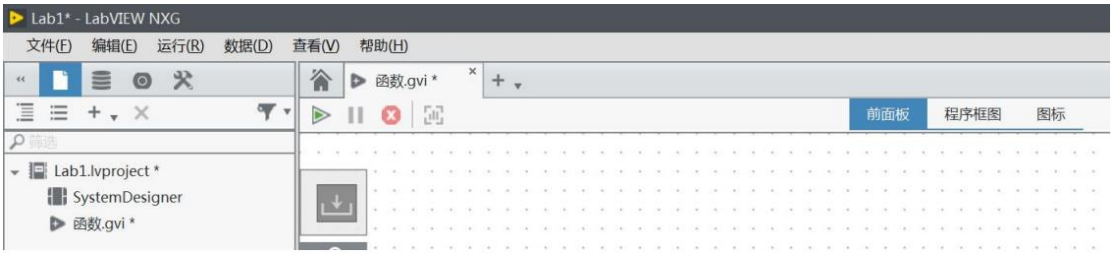

图 4.1 窗口重命名

② while 循环：

While 循环是一个控制流语句，用于重复执行一个函数，直到满足给定的条件终端。条件终端定义循环结束的时间。条件终端有两种设置：如果为 True 则继续和如果为 True 则停止。如果设置为 True 继续，则仅当布尔控件返回 True 时才进行循环。如果设置为 True 则停止，布尔值为 True 时，循环将中止。默认情况下，条件终端是“True 则停止”。

a. While 循环示例

使用 while 循环，产生一组 0 到 10 之间的随机整数，当产生的数与用户给定的数值匹配时，停止产生随机数。

(1) 新建一个 VI；

(2) 在程序框图窗口放置一个 while 循环（左侧程序流模块栏里选取）；

(3) 放置随机数模块产生随机数（数据类型/数值/随机数），产生 0 到 1 之间的随机浮点数；

(4) 为了产生 0 到 10 之间的随机数，放置乘法模块和数值常量输入控件以及最近数取整模块（数据类型/数值）；

(5) 创建一个数值显示输出端口，显示每一次循环产生的随机数：放置一个数值接线端（数据类型/数值），这个接线端默认的行为特性是“控制”（从前面板输入），需要改变其特性为“显示”（输出到前面板），重命名显示端口为 Random Integer；

(6) 创建一个数值输入控制端口（数值接线端），接收匹配的整数。重命名控制端为 Number to match；

(7) 使用“等于？”模块（数学/比较），比较两个数值的大小；

(8) 在前面板窗口创建一个停止按钮（按钮/停止按钮），回到程序框图窗口，在未放置项将停止控件拖放在框图中；

装 订 线

(9) 放置“或”模块（数据类型/布尔），将布尔控制两个输入分别与比较器的输出、停止按钮连接。当输出随机数与输入匹配或者按下停止按钮，循环将中止；

(10)回到前面板，将未放置项拖放进窗口界面。前面板和程序框图如图 4.2 所示：

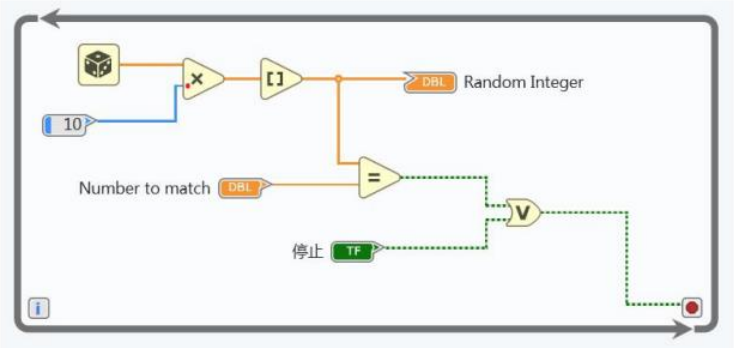

图 4.2 前面板与程序框图

(11)在前面板输入一个匹配数值，运行程序，观察结果；

(12)在程序框图窗口上方菜单条中，点击“高亮显示执行结果”，运行程序的时候，可以观察到程序框图中数据流的变化。

b. 问题 1

对于以下问题，在实验报告中都需要提供程序图和前面板图。

(1) 完成以上示例；

(2) 给 while 循环增加一些延时（比如 1 秒钟），以便能观察到数据的变化；

(3) 增加一个布尔指示器，当随机数大于 5 时打开，小于或等于 5 时关闭；

(4) 增加一个输出端显示循环的次数。

③ 数据类型：

在 LabVIEW 中，每一个对象和连线都跟数据类型有关系。LabVIEW 支持很多数据类型，它们由不同的颜色、形状区分。部分数据类型如图 4.3.1 所示：

a. 示例

下面列举簇数据类型的使用：接收一串字符串，将其反转；对给定的数组元素数值增加偏置量。

装 订 线

图 4.3.1 部分数据类型

(1) 新建一个 VI；

(2) 在前面板创建一个空白簇，如图 4.3.2 所示：

图 4.3.2 新建空白簇

(3) 在空白簇中放置一个字符串输入控件、一个数值输入控件和一个数值数组输入，将数组尺寸改为 5（可通过鼠标拖拉数组外框或直接在属性里修改）。在簇外面再放置一个字符串输入控件和一个数值数组输入，将它们改为显示模式，如图 4.3.3 所示：

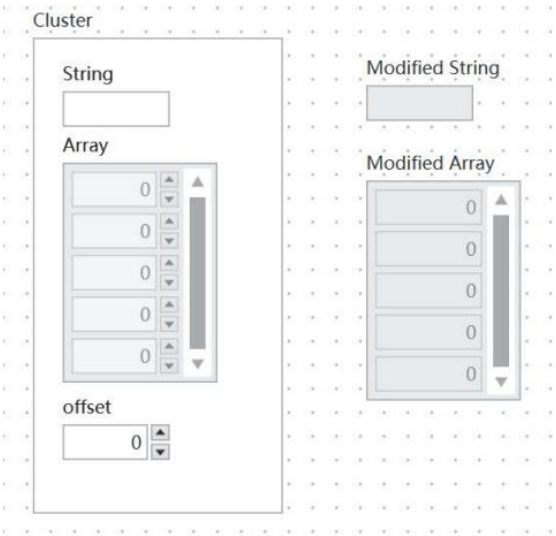

图 4.3.3 显示模式

装 订 线

(4) 在程序框图窗口将未放置项放置在框图界面里，在程序框图中放置簇属性模块（数据类型/簇），以便从簇中读取各个数值。将簇控件与簇属性模块相连，鼠标往下拖拉簇属性模块外框显示更多的属性端口；

(5) 使用反转字符串函数（数据类型/字符串）反转字符串，使用“加”函数（数学/数值）将偏置与数组相加，如图 4.3.4 所示：

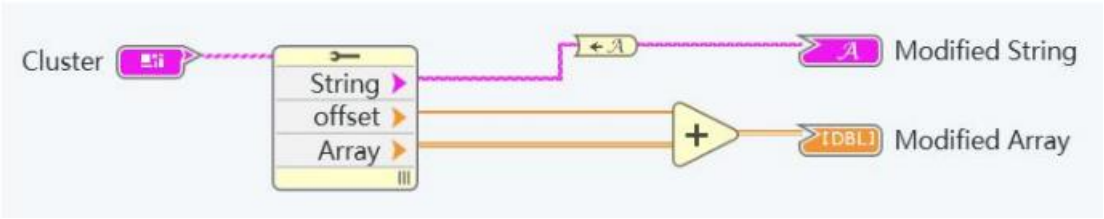

图 4.3.4 反转字符串函数

(6) 在前面板输入控制值，运行程序，通过显示输出确认结果是否正确；

(7) 将改变的值再结合放入簇：复制、黏贴前面板中的簇并将它的属性改为显示。如图 4.3.5 所示：

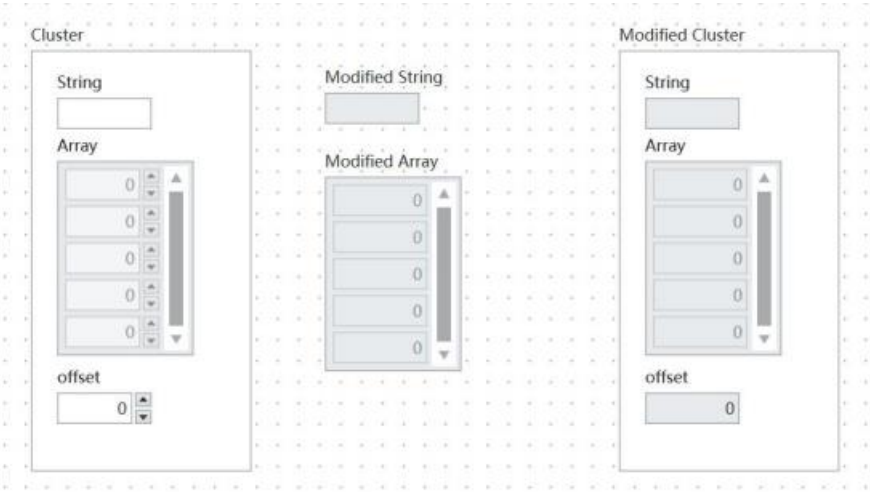

图 4.3.5 将改变的值再结合放入簇

(8) 将新的簇（未放置项）放置在程序框图中，为了将改变的字符串、数组和新的簇结合在一起，需要再次用到簇属性模块。因为要将数据写入簇，所以需要将簇属性设置为“全部为写入”。最终的程序框图如图 4.3.6 所示：

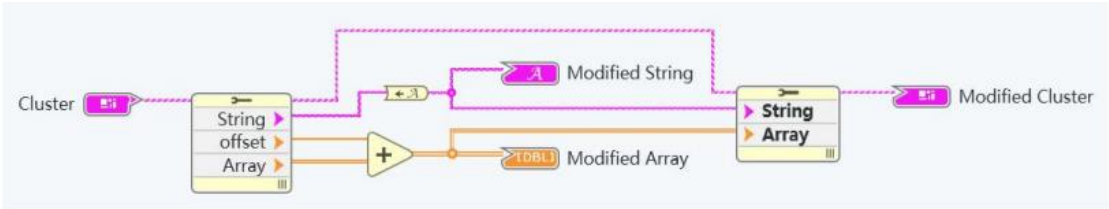

图 4.3.6 最终程序框图

(9) 运行程序并比较结果。

b. 问题 2

(1) 完成上面的示例，在实验报告中提供程序框图和前面板运行结果；

(2) 是否能将一个整型控制输入与双精度类型显示输出相连？整型输入能否与字符串输出相连？如果不可以，需要使用什么模块进行类型转换？

④ 数组

数值中的元素都是相同类型的数据，可以是数值、布尔、路径、字符串、波形和簇。当处理一批类似数据或作重复计算时，可以考虑使用数组。存储波形数据或循环中产生的数据时，使用数组是一个理想选择。

a. 示例

下面举例数值的使用：给定一个数值数组，找出其中的最大元素。

(1) 新建一个空白 VI；

(2) 在前面板添加一个数值数组输入，通过往下拖拉数组外框将元素数量增加到 5。将数值数组输入重命名为“Array”；

(3) 添加一个数值输入控件，将它改为显示模式并重命名为“Maximum value”；

(4) 在前面板添加一个 For 循环，将“Array”终端放在循环外面，“Maximumvalue”终端放在循环里面；

(5) 将数值终端与 For 循环的左边框相连，在边框上会产生一个“隧道”，隧道使得循环与外面的模块能够进行数据通信。隧道有索引使能功能，当它设置为自动索引使能时，输入循环结构的数组元素在每一次循环的时候送入一个元素。如果索引没有使能，循环结构将一次调用数组中的全部元素；

(6) 为了在 For 循环中对输入数组的元素逐个跟设定的最大值进行比较，需要在 For 循环中添加一个“大于？”函数（数学/比较）。再放入一个“选择”模块（程序流）；

(7) 假如当前输入循环的数组元素比最大值大，需要更新最大值。将隧道输出与“大于？”模块的 X 端相连，同时连接到“选择”模块的真值输入端。将“大于？”模块的输出与“选择”模块的选择端连接；

(8) 我们需要存储当前的最大值以备下次循环迭代的时候使用。移位寄存器是将两次相邻迭代联系起来的数据传输通道。将“选择”模块的输出连接到循环的右侧边框线上，会默认建立一个“隧道”，点击鼠标右键，将它改为移位寄存器模式，这时鼠标

装 订 线

光标也变成移位寄存器的图标，点击循环的左边框，完成移位寄存器的放置（移位寄存器的左侧方框可以看作是来自上一次迭代结果的输入，右侧方框是本次迭代的输出）。程序框图如图 4.4.1 所示：

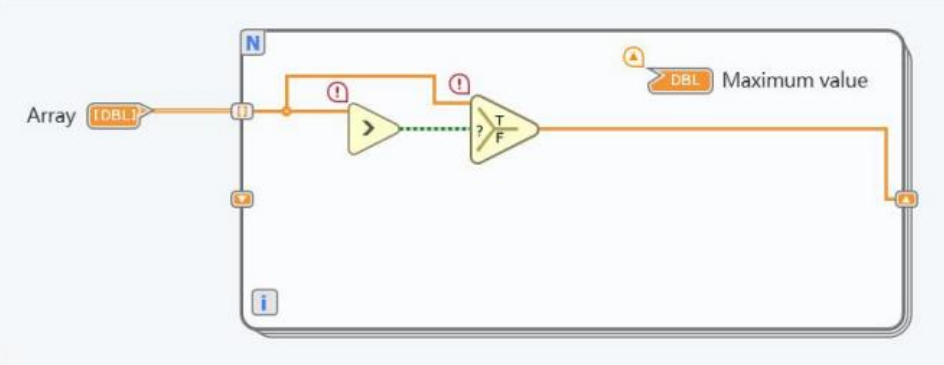

图 4.4.1 程序框图

(9) 移位寄存器的左侧方框是上一次迭代产生的最大值，用来与当前输入的数组元素进行比较。将移位寄存器的左侧方框与“大于？”模块的 Y 端以及“选择”模块的假值连接；

(10) 将“选择”模块的输出与 Maximum value 显示控件相连。完整的程序框图如图 4.4.2 所示：

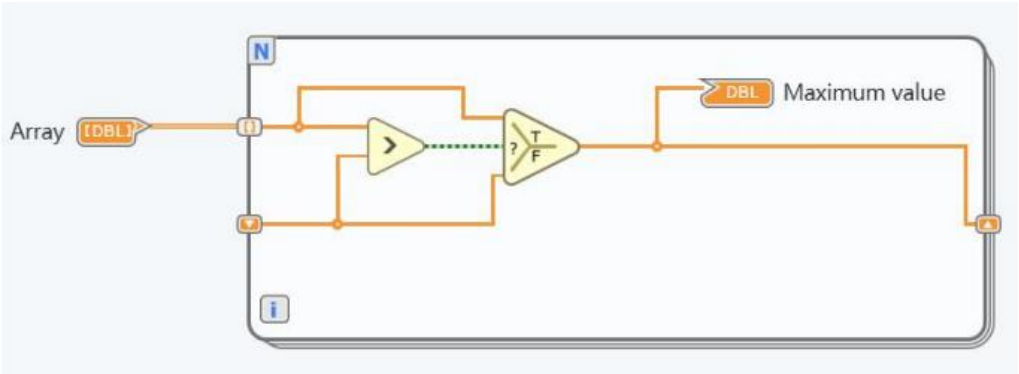

图 4.4.2 完整程序框图

(11) 在前面板给 array 输入数值，运行程序确认是否正确。

b. 问题 3

(1) 完成上面的示例，在实验报告中提供程序框图和前面板运行结果；

(2) 将数组元素最大值改为 0，再次运行程序，还能找到最大值吗？更改例程，使得每运行一次程序都能找到最大值；

(3) 在问题 2 的基础上，更改程序框图，能够得到最大值的索引值（在数组中的位置）。结果可参考下图：

装 订 线

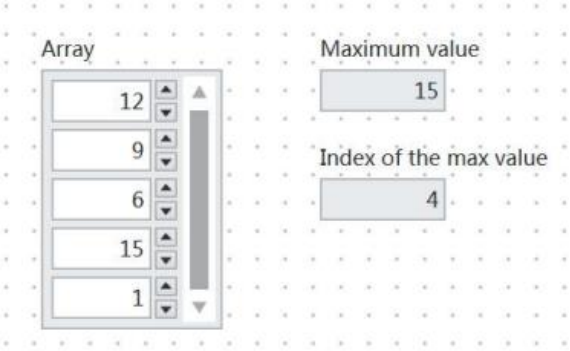

图 4.4.3 结果参考图

(4) 完成以下 VI：产生两个数值范围在-0.5 到 0.5 之间的随机数，比较这两个随机数值的大小，如果第一个数比第二个数大，Team A 得分；如果第二个数比第一个数大，Team B 得分。当其中一个队得分达到 50 时，程序停止运行。显示最终的两个随机数、循环的次数、Team A 和 Team B 的得分以及哪一对赢。前面板如下图所示：

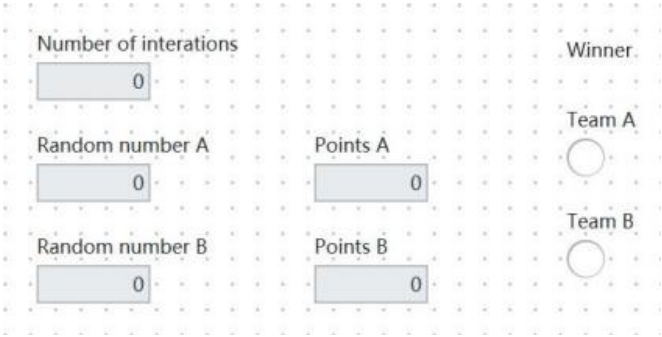

图 4.4.4 前面板参考图

五、实验数据分析与问题回答

① 问题 1：

(1) 示例：

如图 5.1.1 和 5.1.2，分别为程序图和前面版图。

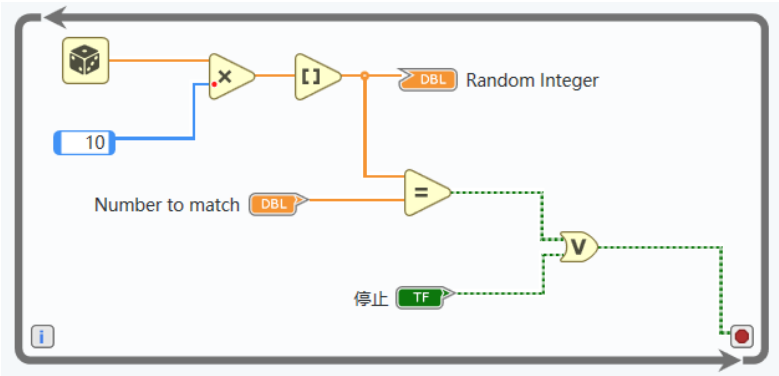

图 5.1.1 程序框图

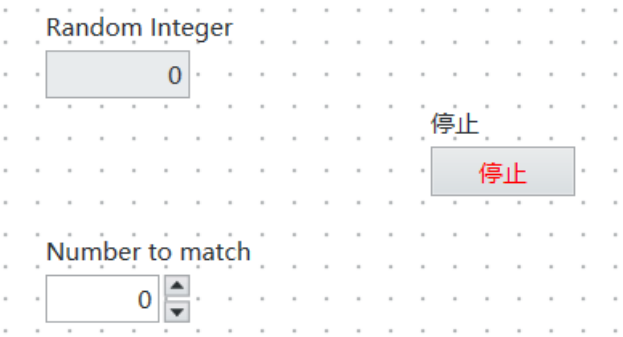

图 5.1.2 前面板图

根据电路逻辑，用户输入需要匹配的数字后，程序将生成 0 到 10 之间的随机数，直到随机数与输入数字相等或用户按下停止键后，程序才停止运行。如图 5.1.3，为程序正常运行的结果图，可以看到结束后随机数与用户输入数字相同，都是 5。

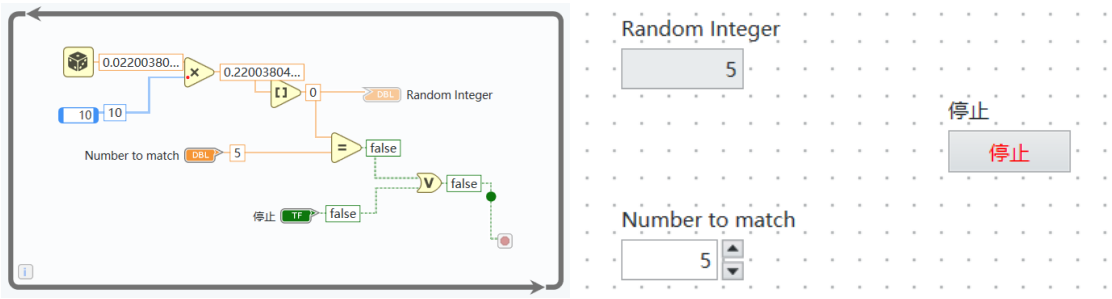

图 5.1.3 随机数匹配

同理，当用户输入数字不在 0 到 10 之间时，程序将无限循环。只有当用户按下停止键后，程序才会结束，如图 5.1.4 所示。

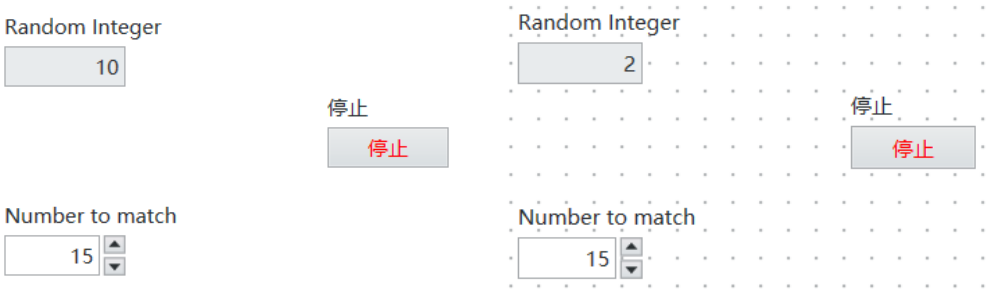

图 5.1.4 按下停止键结束程序

(2) 给 while 循环增加一些延时（比如 1 秒钟），以便能观察到数据的变化：

如图 5.1.5，为了在 while 循环中增加延时，可以添加等待模块（程序流/等待），并设置延时时间为 1000ms。其前面板图如 5.1.6 所示，与未添加等待模块相比并没有产生变化。

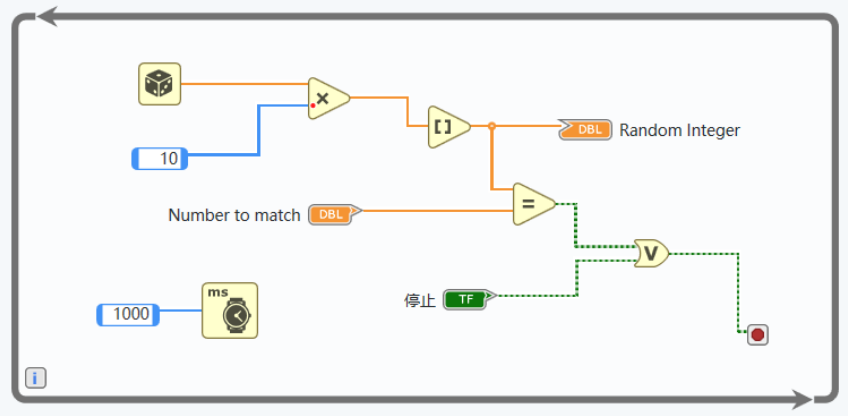

图 5.1.5 程序框图

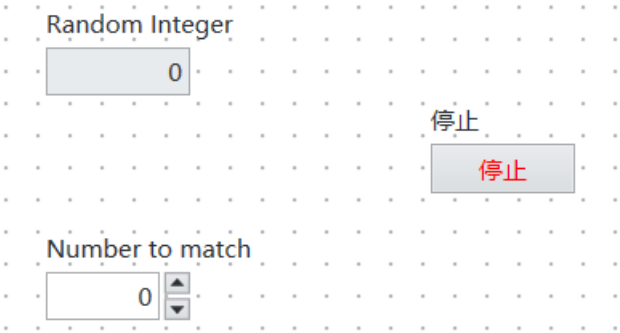

图 5.1.6 前面板图

此时再次运行程序，在前面板中会发现 Random Integer 显示数字改变会出现明显的延时，而程序框图运行一个循环后会停滞一段时间，数据变化更易被观察。

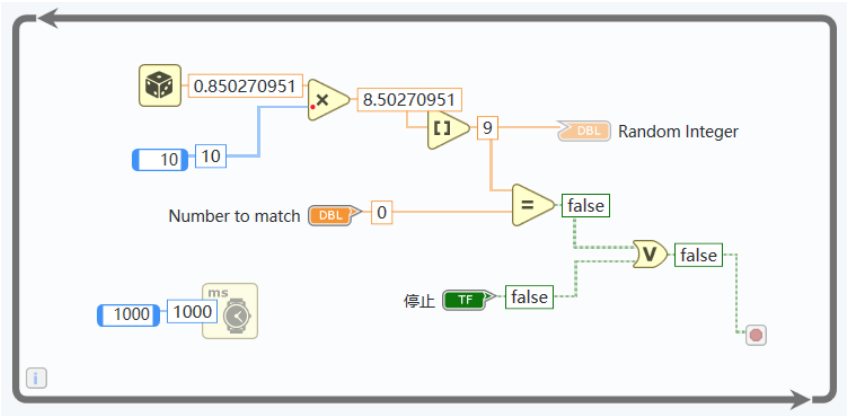

图 5.1.7 延时运行图

(3) 增加一个布尔指示器，当随机数大于 5 时打开，小于或等于 5 时关闭：

如图 5.1.8，可以在程序框图中添加一个比较模块，用于比较随机数与 5 的大小，并将输出连接到布尔接线端。当 5 小于随机数时，输出为 True，则指示器打开；否则输出 false，指示器关闭。其前面板图如图 5.1.9 所示。

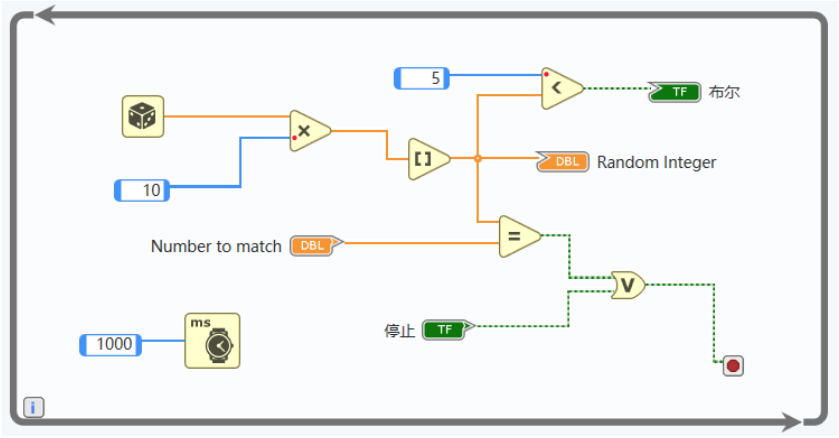

图 5.1.8 程序框图

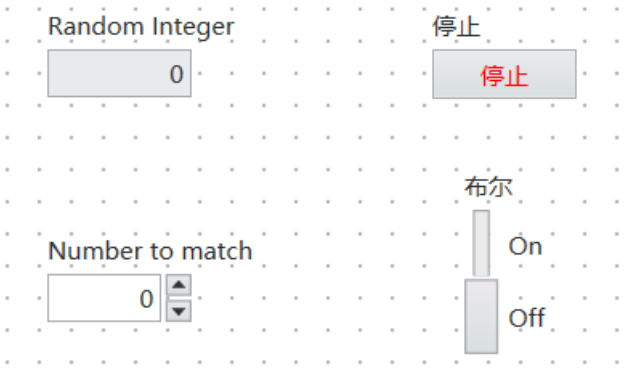

图 5.1.9 前面板图

如图 5.1.10，为程序的运行截图。可以看到，当随机数为3 < 5时，布尔指示器处于关闭状态；当随机数为6 > 5时，指示器打开，符合预期功能。

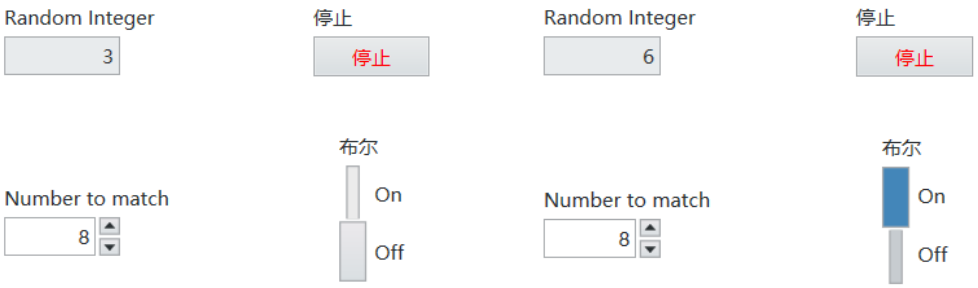

图 5.1.10 布尔指示器

(4) 增加一个输出端显示循环的次数：

显然，在 while 循环中有一个循环计数接线端，保存有循环的次数，将其连接到数值输入控件即可，如图 5.1.11 和图 5.1.12 所示。

装 订 线

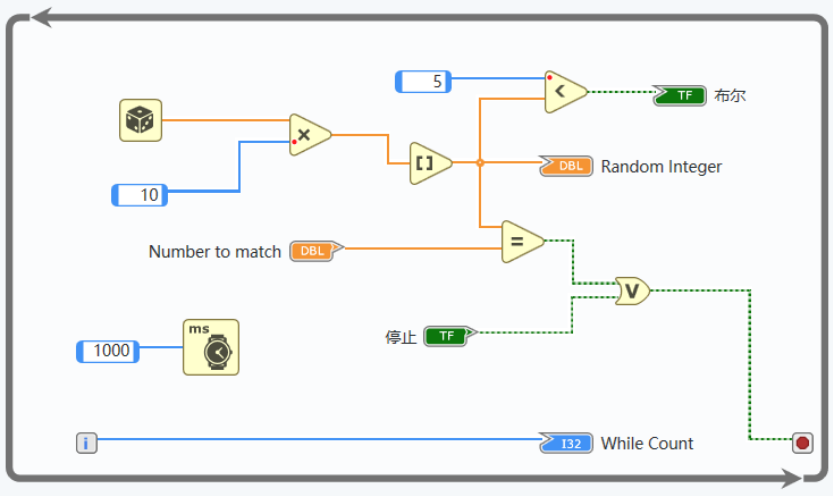

图 5.1.11　程序框图

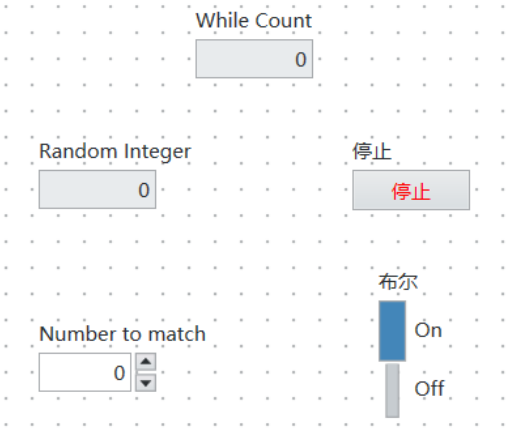

图 5.1.12　前面板图

如图 5.1.13，设置需要匹配的数字为 7，当程序运行结束时（匹配到随机数），总共进行了 25 次循环，功能正确实现。

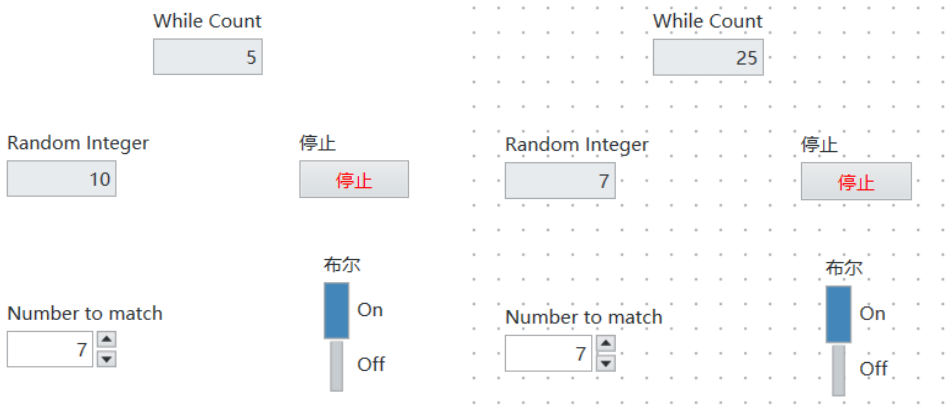

图 5.1.13　显示循环次数

② 问题 2：

(1) 示例：

如图 5.2.1 和图 5.2.2，在前面板中输入控制值，可以看到显示输出中的字符串为输入字符串的反序；输出数组中各个值为输入值加上偏移量，与预期的结果相符合。

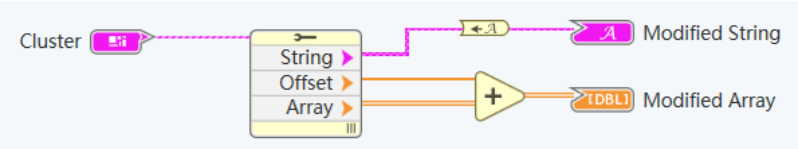

图 5.2.1 程序框图

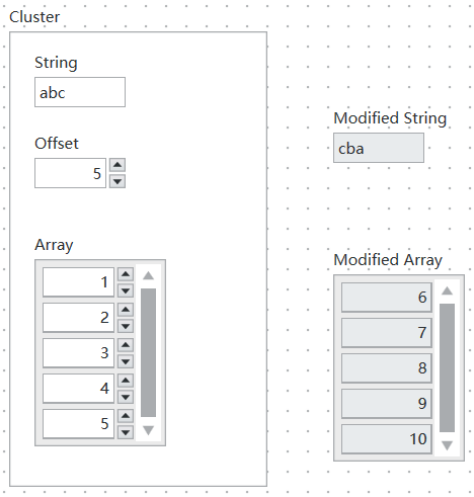

图 5.2.2 前面板图

如图 5.2.3 和图 5.2.4，使用簇显示修改后的输出字符串和数组，其中输出字符串与输入反序，输出数组由输入数组值加上偏移量得到，偏移量显示不变，结果正确。

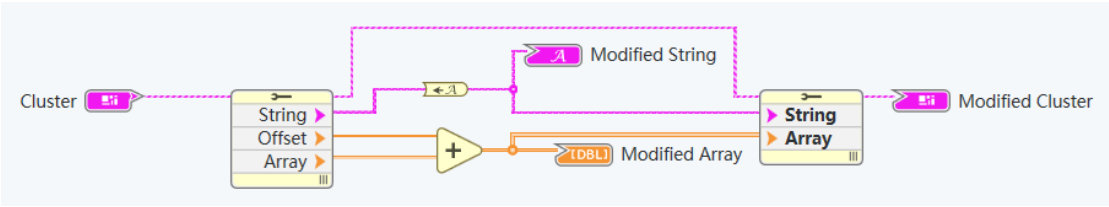

图 5.2.3 最终程序框图

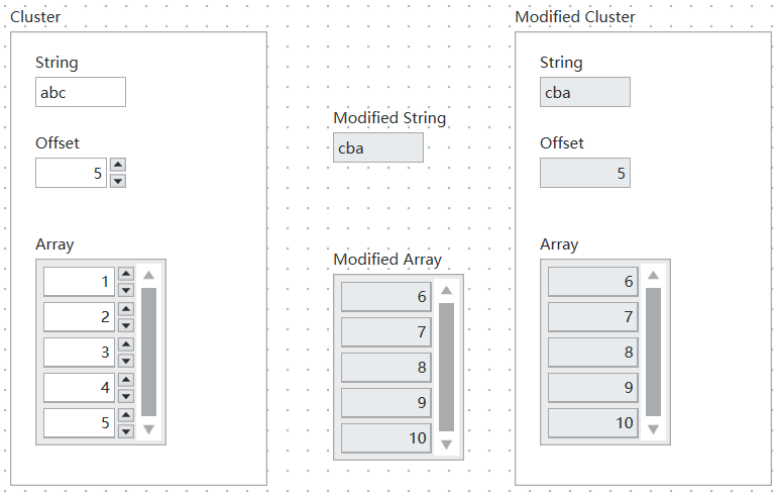

图 5.2.4 最终前面板图

装 订 线

(2) 是否能将一个整型控制输入与双精度类型显示输出相连？整型输入能否与字符串输出相连？如果不可以，需要使用什么模块进行类型转换？

如图 5.2.5，可以将整型控制输入与双精度类型显示输出相连，其在前面板中的显示值与输入相同，如图 5.2.6 所示。

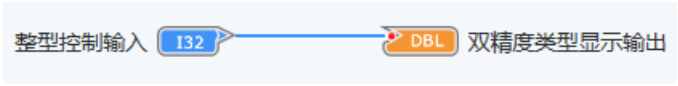

图 5.2.5 程序框图

图 5.2.6 前面板图

然而，当尝试将整型输入与字符串输出相连时，程序出现“类型冲突”的报错，即整型输入不能与字符串输出直接相连，如图 5.2.7 所示。

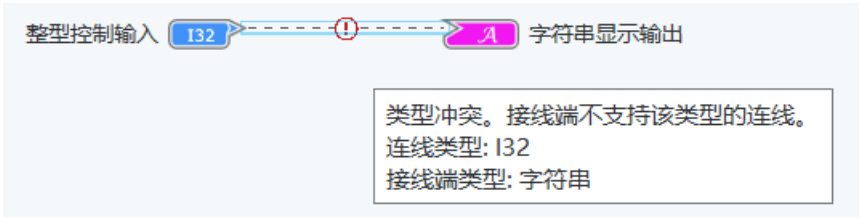

图 5.2.7 类型冲突报错

为了将两者连接，可以使用“数值至十进制数字符串转换”模块，其能够将整型数值转换为对应的字符串，如将整型十进制数 8 转为字符串“8”，从而输出到显示控件，如图 5.2.9 所示。

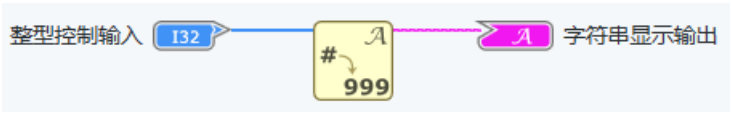

图 5.2.8 数值至字符串转换

图 5.2.9 字符串显示输出

③ 问题 3

(1) 示例：

如图 5.3.1，移位寄存其中保存的是到目前循环为止所有输入数组中的最大数值，

其将会与数组中的数值依次进行比较并更新。如图 5.3.2，对程序进行测试，其能够正确找到输入数组中的最大值 100，功能符合预期。

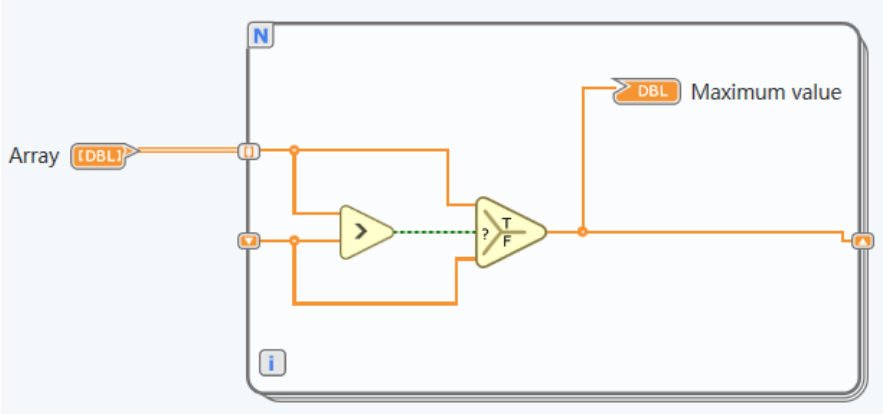

图 5.3.1 程序框图

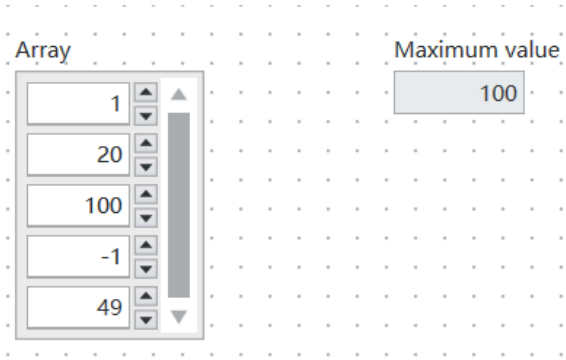

图 5.3.2 前面板图

(2) 将数组元素最大值改为 0，再次运行程序，还能找到最大值吗？更改例程，使得每运行一次程序都能找到最大值：

显然，由于寄存器中保存的最大值并不会因为程序重新运行而清空，当上一次运行结果大于下一次输入数组中的最大值时，程序最终的输出仍会是上一次运行找到的最大值，导致结果出现错误，如下图所示：

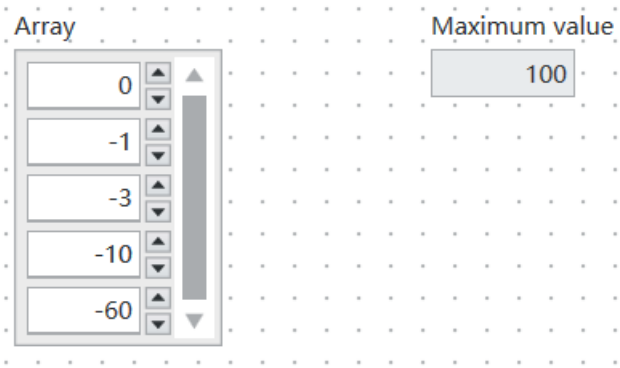

图 5.3.3 找不到最大值

装 订 线

基于此错误产生的原因，我们需要在程序重新运行时将移位寄存器中的值清空。因此，我们调用了索引数组模块（数据类型/数组/索引数组），在程序开始运行时将移位寄存器中的值设置为输入数组的第 1 个元素，即 Array[0]，程序框图如图 5.3.4 所示：

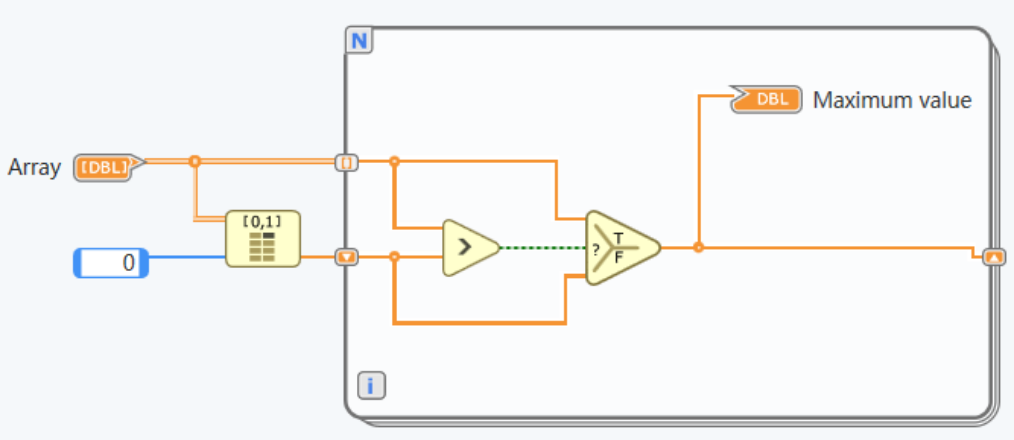

图 5.3.4 程序框图

如图 5.3.5，重新运行程序，可以看到此时能够正确输出最大值 0。在此基础上再次修改输入数组的数值，程序输出为-3，与预期结果相吻合。

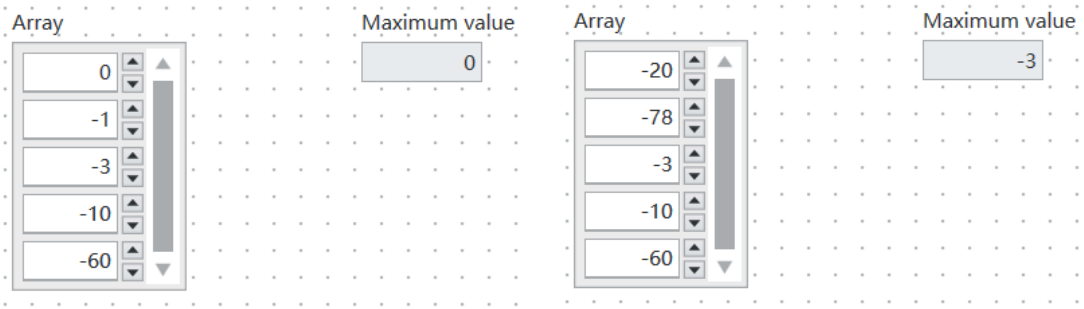

图 5.3.5 前面板图与正确运行结果

(3) 在问题 2 的基础上，更改程序框图，得到最大值的索引值：

为了得到最大值的索引值，我们采用了与示例类似的思想。在问题 2 的基础上，当数组中某一个数比当前保存的最大值更大时，需要将其索引值保存。由于数组索引与循环次数是相同的，因此只需要将此时的循环次数保存；若数组中的数比当前最大值小，则索引值不变。因此，考虑使用“选择”模块对此逻辑进行控制，根据数值比较结果选择当前循环次数或保存的索引值作为输出。当然，为了与问题 2 中的处理保持一致，需要将索引值对应的移位寄存器初始值设置为 0。最终得到的程序框图如图 5.3.6 所示。运行程序，其能够正确输出数组中最大值-3 的索引 2，如图 5.3.7，结果与预期相吻合。

装 订 线

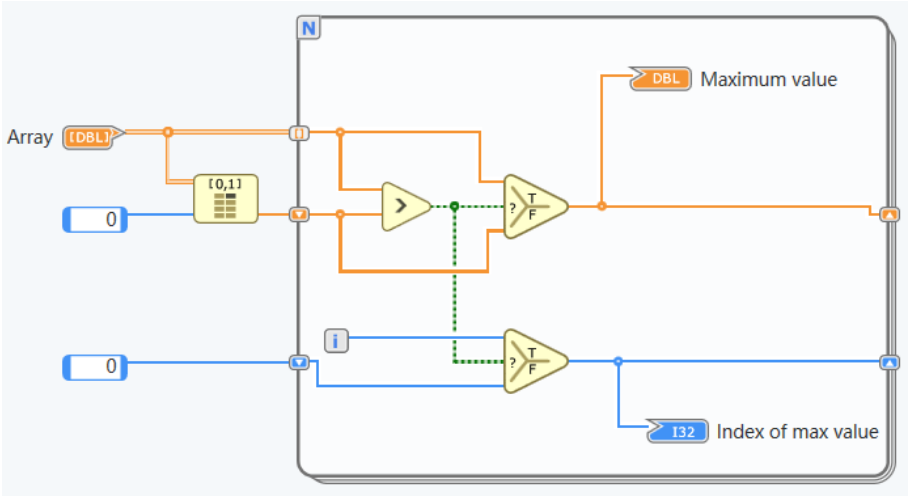

图 5.3.6 程序框图

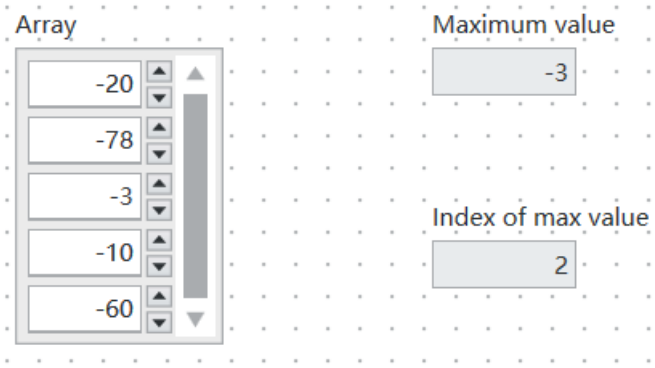

图 5.3.7 前面板图

(4) VI 实现：

如图 5.3.8，为最终设计的程序框图。首先，由于每次循环都需要对上一次 Team A 和 Team B 的分数进行修改（加 1 或保持不变），需要使用移位寄存器保存两队的分数，并将其初始分数赋值为 0。除此之外，在修改分数时，只有加 1 或保持不变（即加 0）两种操作，选择何种操作只与两个随机数的比较结果有关。因此，可以使用“布尔值至整数转换”模块（数据类型/布尔/布尔值至整数转换）将比较产生的布尔值 True/False 转为整数 1/0，再直接与上一次的分数相加。因为 Team A 和 Team B 的分数修改操作始终相反，只需要在框图中加入一个“非”运算对其进行区分。当某一队的分数到达 50 时，将会使“或”运算输出 True，进而使程序停止运行。最后，由于循环次数从 0 开始计数，其始终比实际的迭代次数少 1，需要修正后再行显示。

如图 5.3.9 和图 5.3.10，分别为 Team A 取胜和 Team B 取胜时的程序运行截图，其迭代次数分别为 96（50+46）、94（44+50），VI 效果符合预期。

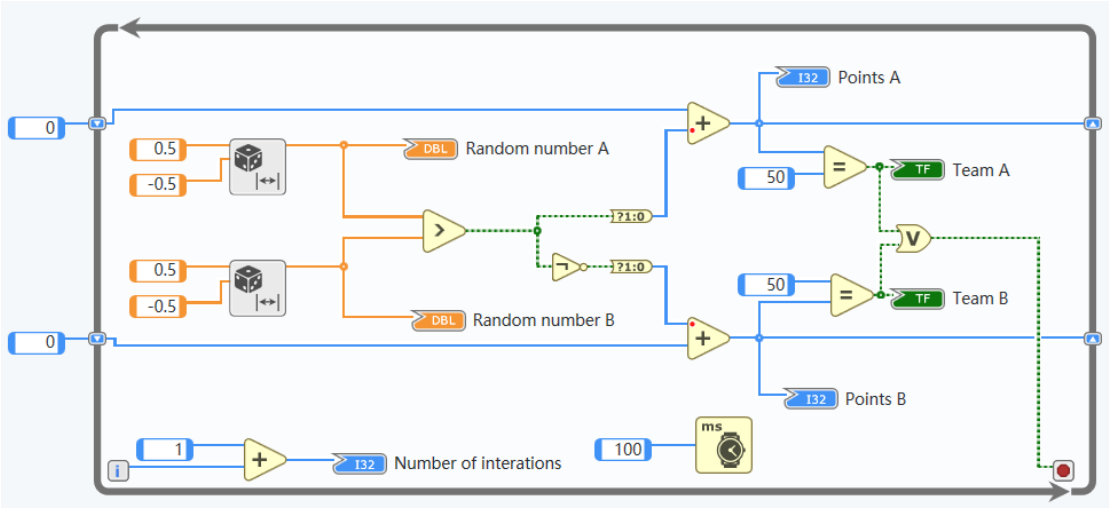

图 5.3.8 程序框图

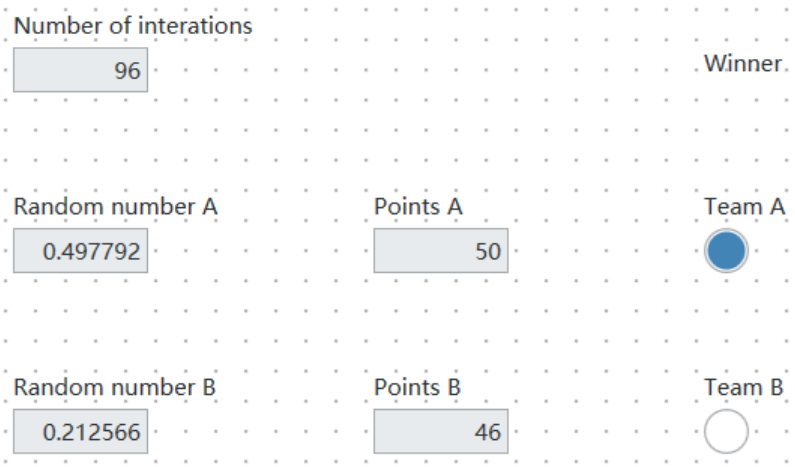

图 5.3.9 程序框图（Team A 取胜）

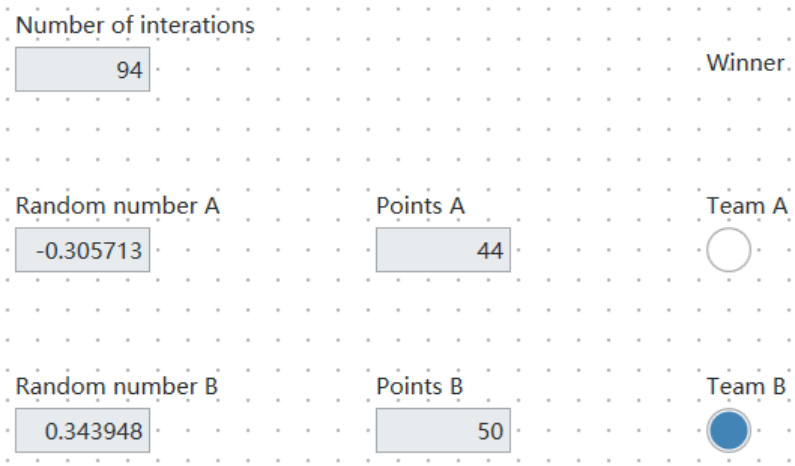

图 5.3.10 程序框图（Team B 取胜）