运动控制产品

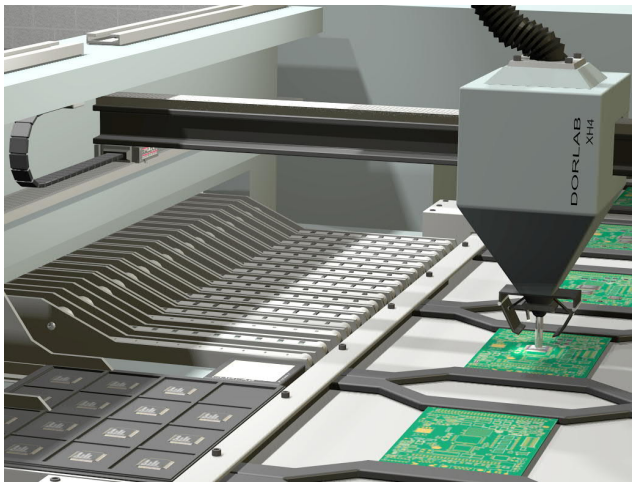
应用说明
拾取和放置示例

AN00179

Rev C (CN)

Mint 是一种易于使用的高级编程语言。它具有丰富的功能，包括编写模块化、结构化的程序。 Mint 语言包括子例程、函数、任务、结构体（用户定义的数据类型）、条件语句、循环语句等。

Mint 还提供了广泛的专用功能，可与 ABB 控制器的硬件连接。它还提供丰富的关键字，大大简化了复杂运动控制的编程。



引言

Mint 是一种顺序语言（即，在之前的代码行执行完成后，才会执行特定的代码行），它还为用户提供并行处理（通过多任务）和事件/中断驱动操作的能力（例如，通过数字输入事件），使语言整体非常灵活，特别适用于机器和运动控制应用。
本文档深入介绍了一些技术，可用于轻松实现如上图所示的拾取和放置动作。

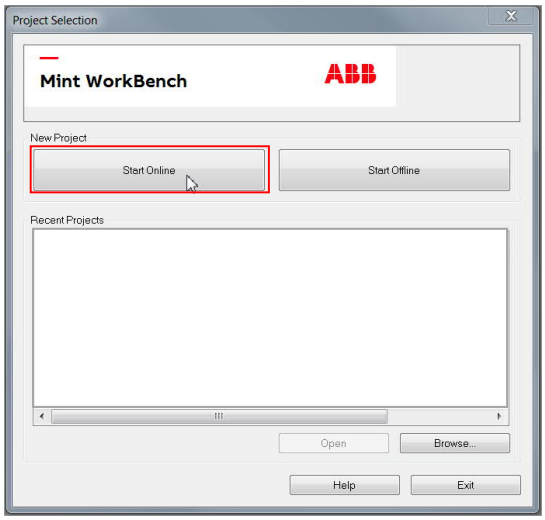
应用

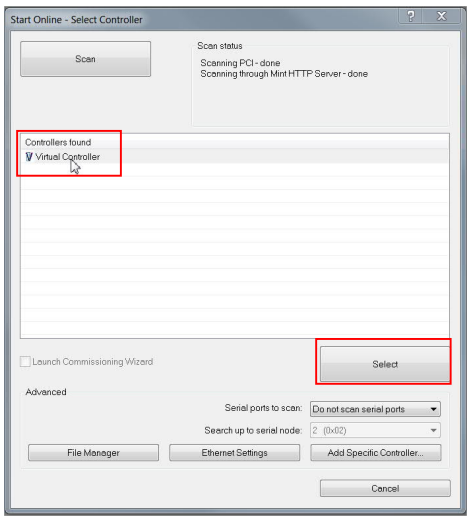
本系统是一个电路板安装型应用。该系统的机械部件通常是龙门或“H 桥”布局。其中两个电机用于控制 Y 轴位置，一个电机控制 X 轴的横向行程。拾取器通常采用气动操作（上/下阀门，真空开/关和夹入/拉出），带有用于 IC 夹持器的棘轮式分度系统（电磁阀驱动）

本文档的配套程序可在 Mint 虚拟控制器（Mint Workbench 的一部分）上运行，在没有任何实际硬件的情况下运行代码并模拟机器动作。但在实际应用中，如果需要，可以在 Nextmove 控制器中执行。示例代码假定有五个 IC 托盘，每个 PCB 仅使用这些设备中的每一种最多五个（但是，可以使用后面描述的 Mint 常量轻松地扩展）。

使用虚拟控制器

启动 Mint Workbench 并单击“联机启动”。
Mint Workbench 会找到一个虚拟控制器。





选中“虚拟控制器”，然后单击“选择”按钮，您就可以在线访问虚拟控制器。虚拟控制器可以控制 16 个虚拟轴和多个虚拟 I/O 组。可以下载 Mint 程序并在虚拟控制器上运行，可以免硬件开发应用程序。甚至可以将主机应用程序连接到虚拟控制器（通过“setVirtualControllerLink”）。

执行“文件|打开”操作，打开随附的 Mint 示例程序。程序将出现在“编辑器”窗口内。要在虚拟控制器上下载并运行它，执行“程序|编译|下载和运行”（或使用键盘快捷键 CTRL + F5）操作。

Mint Workbench 的监视窗口的“监视”选项卡可用于选择受控轴的位置（0 到 4）。在应用程序执行时，您将看到这些轴移动到编程位置。

示例代码

下段内容包含 Mint 程序的结构和功能。

轴的配置

示例应用使用以下轴。

a. X 轴（模拟真实的 X 轴）
b. 主 Y 轴（在实际应用中，它实际上将被配置为**虚拟轴** - 两个物理 Y 轴都跟随该轴。由于两个实轴都跟随相同的信号源，它们都具有相同的延迟，因此相对于彼此没有延迟，从而确保龙门两侧的同步运行）
c. Y1 和 Y2 轴（模拟真实的 Y 轴）
d. I 轴（模拟传送 PCB 的分度传送带）
e. F 轴（模拟把新板移动到输送带的给料轴）

这些轴在 Mint 程序的 Startup 模块中配置。
Mint 关键字 SCALEFACTOR 用于设置每个轴的用户单位。该示例假设电机配有编码器和传动装置，每 mm 对应 1000 个计数。通过将 SCALEFACTOR 设置为 1000，可以把线性轴的用户单位变为 mm。

同样，输送机和给料轴也被认为是旋转电机。转位输送机每 mm 行程对应 300 个计数，给料轴每 mm 行程对应 100 个计数。

由于设定了比例因子，SPEED 和 ACCEL/DECEL 的单位变为 mm/sec 和 mm/sec^2（例如，X 轴按 $5000mm/sec^2$ 加速，在 100ms 内达到 500mm/sec 的速度）。

要将两个 Y 轴线性电机配置为跟随虚拟 Y 轴，我们只需要告诉这些轴它们需要跟随的信号源（需求位置）以及它们跟随的轴/通道（虚拟 Y 轴）。

```
‘ 配置 Y1 和 Y2 轴如何跟随 Y 主轴
MASTERSOURCE ([ _axY1,_axY2])  = _msDEMAND;
MASTERCHANNEL([_axY1,_axY2]) = _axYM;
```

注意：把“;”字符用于“将此设置用于所有列出的轴”的方式。

在轴使能后，可以使用 Mint FOLLOW 关键字告知物理 Y 轴以 1:1 的比率跟随虚拟 Y 轴。

```
‘ 开始跟随
FOLLOW([_axY1,_axY2]) = 1;
```

注：在代码中，只需要为 X 轴和虚拟主 Y 轴发送目标位置，物理 Y 轴将简单地跟随虚拟 Y 轴。

程序常量

为了使代码更易于阅读并允许稍后更新程序，使用了一些常量来定义应用的某些属性。轴编号，IC 托盘的数量和每个板上 IC 的最大数量都定义为常量。通过利用这些常量，只需要改变常量的值，不用在程序中添加很多“硬编码”，就能增加机器的容量。

```
Const _nNoOfChipTypes As Integer = 5
Const _nMaxNoOfChips As Integer = 5
```

变量声明

本应用程序的主要数据存储在三个数组中：

- 每个 PCB 上需要的每个芯片的数量
  需拾取的每个芯片的 XY 坐标（每个芯片一个坐标）
- 需放置的每个芯片的 XY 坐标

将这些数据从包含 PCB 制造信息的主机 PC 下载到应用程序是很常见的（例如，通过 Mint ActiveX 控件）。在本示例中，我们对数组内容进行了硬编码。

```
Dim nChipQtys(1 To _nNoOfChipTypes) As Integer
Dim fPickLocs(0 To 1,1 To _nNoOfChipTypes) As Float
Dim fPlaceLocs(0 To 1,1 To (_nNoOfChipTypes * _nMaxNoOfChips)) As Float
```

拾取和放置数组是二维的，允许把每个操作的 X 和 Y 坐标存储在单个数组中。如 fPickLocs 数组，该数组存储从存放托盘检索 IC 时轴应移动到的 X、Y 坐标值。由于本示例中有五个存放托盘（_nNoOfChipTypes），因此我们需要存储五个单独的坐标（拾取位置在每个托盘末端）。下表说明了数组存储的数据：

	位置 1	位置 2	位置 3	位置 4	位置 5
X 轴	10 [0,1]	10 [0,2]	10 [0,3]	10 [0,4]	10 [0,5]
Y 轴	200 [1,1]	250 [1,2]	300 [1,3]	350 [1,4]	100 [1,5]

在方括号中，显示了如何对数组元素进行寻址（例如，位置 3 上 300 的 Y 轴坐标将是数组元素[1,3]）。通过将第一个维度的数组索引声明为 0 到 1，可以将此索引与轴号匹配。通过将第二个维度的数组索引声明为 1 到_nNoOfChipTypes，我们可以将这些索引与坐标值匹配。

因此，通常每个元素可以表示为（轴编号，位置编号）。因此，我们可以使用基于这两个参数的 For ... .Next 循环遍历数组内容。
If we examine the code for moving to the Pick location in detail, we can see this in operation ...
如果我们详细检查移动到 Pick 位置的代码，我们可以在操作中看到以下内容 ...

```
‘ 移动拾取位置 ...
VECTORA(_axX,_axYM) = fPickLocs(0,i),fPickLocs(1,i)
GO(_axX,_axYM)
Pause IDLE([_axX,_axYM])
```

在此代码中，“ i ”取自“ For ... Next ”循环，并代表每种芯片类型（1 到 5）。VECTORA 命令执行插值定位（在本例中为 X 和 Y 主轴），确保两个轴同步。定位通过 GO 指令触发。我们可以通过 PAUSE IDLE 语句等待定位完成。

布置位置数据采用相同的原理。在这种情况下，第二维中的元素数量由最大可能的芯片类型数量乘以要放置的每个芯片的最大可能数量（即 5 * 5 = 25）决定。

拾取和放置子例程

为了模块化代码，使用代码模块操作拾取头上的电磁阀。使用 Mint 的 Define 关键字来为数字输出定义一个有意义的名称（同样使代码更易于阅读，并且在以后易于修改 - 例如，在变更输出分配时）。

```
Define opPICK = OUTX(0)
Define opGRIP = OUTX(1)
Define opVAC = OUTX(2)
```

下表说明了拾取和放置的操作顺序：

顺序	输出	拾取例程	放置例程
1	opPICK	Pick head down	Pick head down
2	opVAC	Vacuum on	Vacuum off
3	opGRIP	Grip in	Grip out
4	opPICK	Pick head up	Pick head up

详细看一下这个表格，我们可以看到，除真空和夹持器在拾取时被打开（1），在放置时被关闭（0）外，顺序一般是相同的。因此，我们可以使用子例程将公共代码组合在一起，并使用参数将 0 或 1 传递给例程。

```
Sub doPicker (ByVal bGrip As Integer)
opPICK = _on
Wait(100)
opVAC = bGrip
opGRIP = bGrip
Wait(100)
opPICK = _off
Wait(100)
End Sub
```

同样，为了使代码更容易阅读，我们定义了两个 Mint 常量，而不是使用数字 0 和 1。

```
Const _nPlace As Integer      = 0
Const _nPick As Integer       = 1
```

所以，要拾取设备，我们使用

```
doPicker _nPick
```

... 要放置设备，我们使用 ...

```
doPicker _nPlace
```

芯片载体转位子例程

数字输出 3 至 7 用于为电磁阀供电，而电磁阀将 IC 移动到芯片托盘末端。假设 100ms 的脉冲输出足以执行一个操作。
同样，通过子程序，可以使代码模块化，并且可以使用参数从所提供的行号导出要操作的输出。

```
Sub doIndex(ByVal nRow As Integer)
‘ 通过脉冲数字信号对芯片载体进行转位
‘ 与特定行相关的输出
PULSEOUTX(2+nRow) = 100
End Sub
```

PULSEOUTX 指令在指定时间内以毫秒为单位发送数字输出。

输送机转位和进给空白板

在本例中，两个轴都执行相对定位。Mint MOVER 指令被用于此目的。在使用分度输送时，如果进给轴仍在运动中，则不得进行移动。
这种互锁是通过在向转位输送机发送 MOVER 命令之前，等待进给轴完成它正在执行的任何运动（即它处于 IDLE 状态）来实现的。

```
Pause IDLE(_axF)
MOVER(_axI) = 600
GO(_axI)
```

在本例中，分度传送移动 600mm 的距离。

一旦分度输送机完成移动，给料输送机就可以进给一个新的空白板。由于分度输送机在循环中的这一点是静止的，在继续进行拾取和放置操作前，我们不需要等待进给完成其移动。

```
MOVER(_axF) = 650
GO(_axF)
```

给料输送机必须把板材送入 650mm，以将它们放到分度输送带上。

联系我们

要了解更多信息，请联系您的当地的 ABB 代表，或以下一种方式：

new.abb.com/drives/low-voltage-ac/motion
new.abb.com/drives
new.abb.com/channel-partners
new.abb.com/plc