目　　录

一、Analog operations

1、Analog 2's Offset Converter

快速键名：op84, ato2off

信号：任意数的模拟信号输入 <**ain1**> **到** <**ain*N***>

对应每个输入有对应的模拟信号输出 <**aout1**> **到** <**aout*N***>

描述：Analog 2's Offset Converter **用** 16 **位数作为输入。用这种方法，它将模拟输入从补码转换成有正负之分的码。**

每一个输入都有一个相对应的输出，并且每一组输入输出之间都相对独立。因此，每一个独立的模块都可以进行转换成和输入相对应多的输出。转换是对称的，即转换后的码再进行转换就变成源码。

转换示例（用 16 **进制数表示）：**

<ain1>	<aout1>
0000	8000
8000	0000
FFFF	7FFF
7FFF	FFFF
5432	D432
D432	5432

2、Analog Buffer

快速键名：abuffer, abuf

信号：1 **个数字输入（**Enable**）**

任意数目的模拟信号输入或连续数据输入：<ain1> 到 <ain*N*>

对应每个输入的输出：<aout1> 到 <aout*N*>

描述： Analog Buffer **元素在上升沿**<**enable**>**驱动一个给出的输出对应于输入的水平。只要**<**enable**>**是高电平，在输入中任何一个并发的改变将传递到输出。当**<**enable**> **是低电平时，所有的输出将保持不变。每一个输入都有一个相对应的输出，并且每一组输入输出之间都相对独立。**

注意： **虽然** Analog Buffer **元素能够传递连续的数据，在大多数情况下，建议使用** Serial Buffer **元素。模拟信号和数字信号的值会一直保持直到它们被赋予新的值，与它们不同，大多数连续信号是瞬时的，这意味着它们的数据只能临时保持。**Serial Buffer **元素更适合处理这种特性。**

3、Analog DivMod

快速键名：divmod, adiv
信号/参数：1 个模拟输入：<ain>
2 个模拟输入：商<quotient> 和 余数<remainder>
1 个参数：除数<divisor> (*参见* Numeric Formats)

描述： Analog DivMod 元素对它的输入执行取整和取模操作来产生两个输出。因此，如果 <ain> 的值是 5 而 <divisor> 是 2,5 的整数部分值(truncated)被 2 除的结果是 2 (<quotient> = 2) 而 5 mod 2 is 1 (<remainder> = 1)。所有取整操作是无正负之分的，也就是说所有的值都是正的。设置<divisor>到 256d 将返回高位数为<quotient>，返回低位数为<remainder>.

4、Analog Equate

快速键名：equ
信号/参数：1 个模拟输入：<ain>
1 个可选的数字输入：<enable>
任意数目的数字输出：<o1> 到 <o*N*>
每一个输出都有一个对应的参数：<value1> 到 <value*N*> (参见 Numeric Formats)

描述：若 <value> 参数对应的值与输入的模拟信号值相匹配，那么 Analog Equate 元素将使输出信号值为高。输出将保持为高直到另外一个匹配值被发现。

注意：1 个 Analog Equate 元素的输出在产生之前是不会变化的。这表明两个有不同<value>参数的输出有可能同一瞬时同时达到高。

当可选输入 <enable> 为高时，激活此元素;当可选输入 <enable> 为低时，所有输出为 0;每次 <enable> 为高时，此元素就将输入重新求值并赋予相应的输出为高。

5、Analog Flip

快速键名：aflip
信号：任意数目的模拟输入：<ain1> 到 <ain*N*>
每一个输入都有一个对应的模拟输出：<aout1> 到 <aout*N*>
描述：Analog Flip 信号在输出的基础上产生了输入的 2 补码。因此它将从 0%到 100%的范围转化成了 100%到 0%。(1 个输入为 50%的余不变）每一个输入都有一个对应的输出，每一个输入/输出对都是相对独立的。
示例： Analog Flip 信号可用于反转触摸屏的感应信号，或反转用于控制 CPC-CAMI 的感应信号。当 PAN-TILT 单位装配反了，这时，右变成了左，上变成了下。

Sample inversions:

<ain1>	<aout1>
50%	50%
0%	100%
100%	0%
75%	25%
25%	75%
80%	20%
20%	80%

6、Analog Initialize

快捷键名：init

信号/参数：

信号输入的形式：1 **个数字输入：**<**trig1**>

任意数量的模拟输出：<**aout1**>**到**<**aout*N***>

对应于每个输出，都有一个对应的参数：<**value1**>**到**<**value*N***>**（参见** Mumeric Formats）

信号输出的形式：任意数量的数字输入：<**trig1**>**到**<**trig*N***>

1 **个模拟输出：**<**aout1**>

对应于每个输入，都有一个对应的参数：<**value1**>**到**<**value*N***>**（参见** Mumeric Formats）

描述：在单一的输入形式中，Analog Initialize **信号通过它对应于在输入信号的每一个上升沿的**<**value**>**参数来驱动每一个输出到一个特定的值。**

在单一输出形式中，信号将对在任何一个输入的上升沿对输出值进行初始化。输出将被设置成对应于最近升高的输入的<**value**>**参数。**

在启动时，所有的输出都为 0 **值，但当输入仅有一个且被赋予信号名** 1 **时除外。在这种情况下，输出将通过它们对应的**<**value**>**参数而获得特定的值。**

7、Analog Integral

快速键名：Integral

信号/参数：1 **个模拟输入：**<**ain**>

1 **个模拟输出：**<**aout**>

1 **个双精度参数：**<**ramp_time**>**（参见** Numeric Formats）

描述：Analog Integral **信号产生了** 1 **个根据输入按比例改变的输出信号，例如：当输入为** 100%，**输出值在被**<**ramp_time**>**指定期间就在** 50%**到** 100%**之间浮动，而当输入为** 0%**时，输出值在同一期间段内在** 505 **到** 0%**之间浮动。当输入为** 50%，**输出保持其现有值不变。**

输出值在 50%到 100%之间转化的时间由以下公式计算：

输出值转化时间=(<ain> - 50%) / 100% * <ramp_time> <ramp_time>：浮动时间

输出值在 50%到 0%之间转化的时间由以下公式计算：

输出值转化时间=(50% - <ain>) / 100% * <ramp_time>

当输入值为 75%，<ramp_time>为 5 秒时，输出值的转化时间将为 2.5 秒，或为 (75% - 50%) / 100% * 5。

示例：当 1 个速率控制器比如 1 个操纵杆或 spring-return slider 必须提供组件例如摄象机的位置控制时就要用到 Analog Integral 信号。摄象机将在操纵杆回弹到 50%位置时按操纵杆比例移动并锁定它自己的位置。

8、Analog Preset

快速键名：Preset

信号/参数：1 个数字输入：<trig>

任意数量的模拟输出：<aout1>到<aout*N*>

每一个输出都对应有一个目的单元格值：<level1>到<level*N*>（参见 Numeric Formats）

1 个双精度参数：<ramp_time>（参见 Numeric Formats）

描述：Analog Preset 信号在被<ramp_time>指定期间，通过从它现有的值到新值之间进行平滑的浮动来驱动每一个模拟输出信号到它相应的<level>目的单元格值。浮动从<trig>的上升沿开始。所有的输出将同时到达它们最终的水平。

注意：在转化期间，输入的第二个上升沿将驱动输出值立即达到它们的最终值。这叫做“ CUT ”。既然模拟信号能够持有多重合法来源，而且通常是合适的，那么它就可以联合模拟预设的输出与模拟（Analog Ramp）信号的输出并对两者进行比较。当完成后，哪个信号（预设或 Ramp）最后驱动模拟输出值，这个模拟输出值就由此信号决定。在灯光和音量的例子中，由手动控制和预设控制结合起来进行控制。

参见 Analog Variable Preset（模拟预设变量），Analog Ramp

9、Analog Ramp

快速键名：Ramp

信号/参数：2 个数字输入：<up>和<down>

1 个附加数字输入：<mute>

1 个模拟输入：<aout>

1 个双精度参数：<ramp_time>（参见 Numeric Formats）

描述：无论输入<up> 或 <down>为高时，Analog Ramp 信号产生出 1 个线性变化模拟输出信号。<ramp_time>参数指定了开始使输出在 0%到 100%（或 vice-versa）之间浮动的时间。附加<mute>输入使输出在<mute>的每一个上升沿变成 0%，并在下降沿返回原值。如果<mute>信号要保持输出为 0%，它就应由一个锁定信号（Toggle symbol）来驱动。

注意：既然<up>和<down>的输入会忽略<mute>并开始第二个浮动操作，这表明当<mute>为高时要用一个缓冲器来禁止<up>和<down>。

参见 Analog Non-Volatile Ramp

10、Analog Rate Limiter

快速键名：slew, alimit, arl
信号/参数：1 个模拟输入：<**ain**>
1 个模拟输出：<**aout**>
1 个双精度参数：<**ramp_time**>（参见 Numeric Formats）
描述：，Analog Rate Limiter 信号通过一个步进输入或任何与瞬变对应的输入而产生 1 个平滑变化的输出，例如从触摸屏上的滑动条。<**ramp_time**>参数指定了 *slew-rate（回转速率）*，或指定了输出从其现值浮动到新值的时间。例如，如果输出现在为 75%，而<**ramp_time**>等于 5 秒，<**ain**>变成 25%时输出将在 5 秒内从 75%转化成 25%。事实上，指定的时间代表了它开始全范围（0-65535）浮动的时间，并由任何局部浮动来进行衡量。例如，如果时间被指定为 10 秒，从 25%到 75%的转化将需要 5 秒的时间，由一半范围起算。

11、Analog Scaler

快速键名：mxb, ascale
信号/参数：1 个模拟输入：<**ain**>
1 个模拟输出：<**aout**>
2 个参数：<**span**>和<**offset**>（参见 Numeric Formats）
1 个可选参数：<**divisor**>
描述：当<**span**>表示比例因子，并且<**offset**>为最小值时，Analog Scaler 信号用以下公式定义其模拟输入信号的值：

```
<aout> = (<ain> * <span> / <divisor>) + <offset>
```

可选参数<divisor>用来限定大的范围并且默认值为 100%。（既然达不到 100%，当<divisor>未定时，输出就可以被定义的更小而不是更大）。

如果输入达到 0%，不论<offset>的值是多少，输出将立即被设为 0。Analog Scaler without Zero Pass 信号则失去了这种“零传递”特性。

示例：1、CNVCP-2（或 3）音量控制卡（在理想的情况下）有一个大的分贝范围从+14 (100%) 到 -76 dB (0%)。一个更理想的范围是从 0 到-30 db。在这种情况下，<**span**>可以被设在 31%而<**offset**>在 46%。
2、一个模拟输入必须被限定在 0-65535 到 0-25 之间。这个范围或叫做跨度，总共有 26 个值，因此<**span**> = 26。当输入和输出都从 0 开始时，<**offset**>的值也为 0。
3、一个模拟输入需要被限定在 0-10 到 0-30 之间。在此，范围必须被扩大，因此<**span**> = 3 而<**divisor**> = 1。这将让输入乘以 3/1 或 3。当输入和输出都从 0 开始时，<**offset**>为 0。
参见 Analog Scaler without Zero Pass

12、Analog Scaler without Zero Pass

快速键名：mxbz, ascale0
信号/参数：1 个模拟输入：<**ain**>
1 个模拟输出：<**aout**>
2 个参数：<**span**> 和 <**offset**>（参见 Numeric Formats）
1 个可选参数：<**divisor**>
描述：Analog Scaler without Zero Pass 信号除了没有“零传递”特性外，和 Analog Scaler 信号的作用一样。也就是说，当输入为 0%时，输出等于<**offset**>而不会减弱。
参见 Analog Scaler

13、Analog Scaler Buffer

快速键名：mbuffer, asbuffer, asbuf
信号/参数：1 个模拟输入：<**scale_factor**>
任意数量的模拟输入：<**ain1**> 到 <**ain*N***>
对应每个输入的输出：<**aout1**> 到 <**aout*N***>
描述：Analog Scaling Buffer 信号按比例限定它的输出为由<**scale_factor**>定义的范围。也就是说，如果<**scale_factor**> = 50%，输出将被设为对应输入的 50%。（输入为 60%，输出就为 30%。）每一个输入信号都有一个对应的输出，并且每一个输入/输出对之间是相对独立的。
注意：因为<**scale_factor**>是一个范围从 0%到 100%的百分数，模拟输出信号的值就不可能超过它的输入。

14、Analog Scaler Buffer about 50%

快速键名：mbuffer2, asbuffer50, asbuf50
信号/参数：1 个模拟输入：<**scale_factor**>
任意数量的模拟输入：<**ain1**> 到 <**ain*N***>
对应每一个输入的输出：<**aout1**>到<**aout*N***>
描述：Analog Scaling Buffer about 50%信号通过<**gain**>限定它的输入范围在 50%上下。此过程按照以下公式进行：

minimum value of <aout1> = 50% - (<scale_factor> / 2)

maximum value of <**aout1**> = 50% + (<**scale_factor**> / 2)

因此，如果<**scale_factor**>被设为 50%，一个通常在 0%到 100%之间变化的输入信号将被限定在 25%--75%之间。这个特性也许可以用于限制一个镜头在高度放大时面板的灵敏度。一个滑动条将显示从 0%到 100%的等级，但是镜头放大的输出将只会在 25%到 75%之间。

一个联系<**aout1**> 到 <**ain1**>通式为：

<aout1> = (<scale_factor>*<ain1>) + <aout1>min

15、Analog Step

快速键名：astep

信号/参数：2 个数字输入：<**step**> 和 <**reset**>
1 个模拟输出：<**aout**>
任意数量的参数：<**value**>（参见 umeric Formats）

描述：Analog Step 信号在<**step**>的每一个上升沿设定它的输出为下一个指定的<**value**>。当输出接近最后的<**value**>时，它就复位到第一个<**value**>。当<**reset**>升高，输出复位到第一个<**value**>。

16、Analog Sum

快速键名：asum
信号/参数：任意数量的模拟输入：<**ain1**>到<**ain*N***>
1 个模拟输出：<**aout**>
描述：Analog Sum 信号以 16 位整数的形式产生它的输出为它的输入之和。而且无论任何输入改变时，都更新输出值。对于一个或更多加数来说，减少可能是由于用二补码符号来表示负值所致。
与 Analog Flip **对照**

17、Analog to Digital

快速键名：A/D, atod
信号/参数：1 个模拟输入：<**ain**>
最多 16 个数字输出：：<**bit16 (msb)**>, <**bit15**> 到<**bit2**>, 以及 <**bit1 (lsb)**>
描述：Analog to Digital 信号是一个转换器，它把模拟信号转换成数字信号。当低对应于 0 而高对应于 1 时，16 位的模拟输入的值由最多 16 个数字输出来表示（开始于最重要的位，或 MSB），0 表示低电平，1 表示高电平。
如果定义少于 16 个输出，则只输出高位。
参见 Digital to Analog

18、Analog to Floating Point

快速键名：cnet12ieeecnet12ieee
信号/参数：2 个模拟输入：<**whole**>和<**fraction**>
1 个数字输入：<**sign**>
4 个模拟输出：<**byte1 (msb)**>, <**byte2**>, <**byte3**> 和 <**byte 4 (lsb)**>
描述：Analog to Floating Point 信号激活了 Crestron 控制系统和 CSI HVAC 单位之间的通讯。也就是说，它将它的模拟输入值转换为 CSI 单位需要的 IEEE 浮点格式。相反的，Floating Point to Analog 信号通过 CSI HVAC 系统将产生的浮点数转换回模拟值。

<whole>输入是一个 16 位整数；而<fraction>是一个 16 位二进制小数，等于<whole>/65535。因此，如果<whole>等于 32767，<fraction>就等于 32767/65535，或是 0.5。这些输入产生了一个由 4 个输出表现的 4 字节（32 位）IEEE 浮点数，由最重要的位<byte1>开始，

如果输入为负数，<sign>输入为高（真）。

示例：要传送一个小数如 0.49，这个值必须被转换成 16 位小数。Analog Scaler 信号用来限定输入值的范围达到期望的精度级别。例如一个精确到 0.01 的数（如 0.49），则为 65535（模拟信号的最大值）， 为 100，<offset>为 0。（精度为 0.1 时， 必须是 10；精度为 0.001 时， 必须为 1000；精度为 0.0001 时， 必须为 10000。）

以上的计算随后将成为 49 * (65535/100 + 0) = 32112，即 IEEE 传送的正确格式（32112 / 65535 = 0.49）。

参见 Floating Point to Analog。

19、Analog to Indirect Text

快速键名：dpm

信号/参数：1 个模拟输入：<ain>

1 个可选数字输入：<enable>

3 个参数：<Net/Gateway ID>, <field> 和 <format>

1 个可选参数：<RF ID>

描述：Analog to Indirect Text 信号用于在一个或更多触摸面板上显示模拟值。在大多数应用中，这种信号是多余的，因为数字规格能够在 VT Pro-e 中被创造出来。

当把模拟值传送到无线面板时，<Net/Gateway ID>参数为触摸面板的 16 进制的 Cresnet ID。一个 FFh 的<Net/Gateway ID>将传送此值到所有的触摸面板。

当把模拟值传送到双向（2-way）无线面板（例如 STX-3500C）时，<Net/Gateway ID>就指定 *Gateway* （例如 CNRFGWX）的 16 进制的 Cresnet ID。在这些应用中，可选参数<RF ID>必须被用于指定面板的 16 进制的 RF ID。

<field>参数指定了分配在 VT Pro-e 中的间接文本区域。

<format>参数是一个两位数，它提供值的显示格式。第一位数提供小数点的位置，如果需要，第二位数提供数字的数量。（如果第一个<format>数为 0，就不用小数点。如果它是 1，小数点就将位于最右边的位置。如果它是 2，小数点在最右边的第二位，依此类推。）

可选输入<enable>在每一个<enable>的上升沿对显示值进行更新（如果输入改变）。

20、Analog Value Sample

快速键名：sample

信号/参数：1 个数字输入：<sample_all>

1 个可选数字输入：<sample_changed>

任意数量的模拟输入：<ain1>到<ain*N*>

对应每一个输入，都有一个模拟输出：<aout1> 到 <aout*N*>

描述：Analog Value Sample 信号仅只在<sample_all>的上升沿赋予输出以对应于输入的值，其它情况下无论<sample_all>的水平如何，输入怎样变化都不对输出产生影响。这基本上形成了一个样例。每一个输入都有一个对应的输出，并且每一个输入/输出对之间是相对独立的。

可选输入<sample_changed>只反映改变了的信号。

示例：一个 Analog Value Sample 信号占用了一个或多个模拟水平的一个“快照”。在更新的模拟信号通过 Intersystem Communications 信号发到远程系统时，它经常与 Oscillator 信号（震荡器信号）一起用于控制速率。具有代表性的是，当较慢的 Oscillator 驱动<sample_all>去补偿传送错误时，较快的 Oscillator 就驱动可选输入<sample_changed>去检测改变、远程启动系统等。

参见 Oscillator，Intersvstem Communications。

21、AnAnalog Variable Preset

快速键名：presetv

信号/参数：1 个模拟输入：<ramp_time>

任意数量的模拟输入：<level1> 到 <level*M*>

任意数量的数字输入：<scene1> 到 <scene*N*>

对应每一个输入从<level1>到<level*M*>，都有一个模拟输出：<zone1>到 <zone*M*>

1 个数字输出：<busy>

描述：Analog Variable Preset 信号，经常与 Analog RAM 信号一起使用，用于激活 end-user 来定义系统的预设。这与 Analog Preset 信号有所不同，在<level> 和 <ramp_time>里，Analog Preset 信号为在动态时间内可改变的模拟值。（在所有的模拟信号中， <ramp_time>输入为 1 个单精度值。）每一个<level>输入都有一个对应的<zone>输出，并且每一个输入/输出对之间是相对独立的。

Analog Variable Preset 信号在每一个<scene>输入的上升沿使每一个<zone>输出跳变到它的对应<ramp_time>输入的水平。输出在由<ramp_time>限定的时间内从现值跳跃到一个新值。当输出跳变时，<busy>输出升高，并可以当作是反馈信号。在启动时，所有的<zone>输出都被设为 0，<busy>输出为低。

注意：在转变过程中，<scene>输入的第二个上升沿将驱动<zone>输出立即达到它们的最终值。这叫做 cut。

参见 Analog RAM，与 Analog Preset 对照。

22、Decade

信号/参数：1 个模拟输入：<ain>

1 个可选数字输入：<enable>

最高 40 个数字输出：<units0-9>, <tens0-9>, <hundreds0-9>, 和 <thousands0-9>

描述：Decade 信号将它的模拟输入信号转换到数字输出组中，每一位数以十进制数来表示。每一组担任解码器的十分之一。例如，给输入为 125，接着的输出将升高：<hundreds1>, <tens2> 和 <units5>。

可选输入<enable>在值为高时激活信号，在值为低时设置所有的输出为 0。在的模拟输入值可能连续改变的应用中，如从一个 Analog Ramp 信号改变，<enable>能变成一个脉冲信号，以产生期望中的输出。

Decade 信号顺次求出整数部分的值（从最大值开始），就像输出能被用于发送串行数据到 RS-232 口，例如，没有附加逻辑或延迟的需要。在以上的例子中，<hundreds>输出将首先升高，然后是<tens>，最后是<units>。

23、Digital Sum

快速键名：dsum
信号/参数：任意数量的数字输入：<**i1**> **到**<**i*N***>
1 **个模拟输出：**<**aout**>
描述：Digital Sum **信号根据它的输出产生一定数量的高电平输出信号，并更新输出值。**

24、Digital to Analog

快速键名：bda, dtoa
信号/参数：最高 16 **个数字输入：**<**bit16 (msb)**>, <**bit15**> **到** <**bit2**>， **和**<**bit1 (lsb)**>
1 **个模拟输出：**<**aout**>
描述：Digital to Analog **信号是一个数字-模拟转换器。也就是说，它通过它的数字输入驱动它的输出达到预期的值。每一个输入对应一位（开始于最高位，或** MSB**），低电平对应** 0**，高电平对应** 1**。任何不明确的输入被假定为** 0**。**
例如，如果用到了 4 **个输入，**<**bit16**> **到** <**bit13**>**，输出范围就为从** 0%**到** 93.751%**。**
对照 Digital to Scaled Analog，Analog to Digital。

25、Digital to Scaled Analog

快速键名：d/a, dtosa
信号/参数：最高 16 **个数字输入：**<**bit16 (msb)**>, <**bit15**> **到** <**bit2**>**，和** <**bit1 (lsb)**>
1 **个模拟输出：**<**aout**>
描述：Digital to Scaled Analog **信号将它的数字输入转换成一个有范围的模拟输出信号，如下：每一个输入表示一位（从最高位开始，或** MSB**），低电平对应** 0**，高电平对应** 1**。因而得到的输出值范围就由输入定义的二进制值限定，这样当所有定义的输入为高时，输出值为** 100%**，而当所有定义的输入为低时，输出值为** 0%**。**
例如，如果数字 7 **由** 4 **个输入表示，**<**bit16**>=0**（低）而**<**bit15**>**到**<**bit13**>=1**（高）。因为一个** 4 **位数可以有从** 0 **到** 15 **之间的值，而** 7 **是中间值，模拟输出将为** 50%**。**
注意：如果所有 16 **个输入都用到的话，**Digital to Scaled Analog **信号将立即把数字信号转换为模拟信号。**
参见 Digital to Analog **，**Analog to Digital。

26、Floating Point to Analog

快速键名：ieee2cnet1
信号/参数：4 **个模拟输入：**<**byte1 (msb)**>, <**byte2**>, <**byte3**>**和**<**byte4 (lsb)**>
2 **个模拟输出：**<**whole**> **和** <**fraction**>
1 **个数字输入：**<**sign**>
描述：Floating Point to Analog **信号激活了** CSI HVAC **单位和** Crestron **控制系统之间的通讯。也就是说，它将由** CSI **单位产生的** IEEE **浮点数转换成模拟值。相反的，**Analog to Floating Point

信号是将模拟值转换成 IEEE 浮点数。

4 字节（32 位）的 IEEE 浮点数由 4 个输入来表示，由最高字节<byte1>开始。输出<whole>是一个 16 位整数值，表示小数点左边的部分，而<fraction>是一个 16 位二进制小数，等于<whole>/65535。因而，如果<whole>=32767，<fraction>就等于 32767/65535，或 0.5。这个数可以通过 Analog Scaler 信号来达到一个期望中的精度。

输出<whole> 和 <fraction>总是表示浮点数的绝对值。如果浮点数的范围超过了可以表示的范围，每一个输出就被设为它们的最大值（065535）。

要将输出的值限定为从 0 到 10（精度为 0.1），Analog Scaler 就必须有一个 =11 和一个<offset>=0。一个值为 32767 的输入将给出一个值为 5 的输出，这个输出将被看作是 0.5。

要限定输出值为从 0 到 100（精度为 0.01），必须等于 101 而<offset>等于 0。一个值为 32767 的输入将给出一个值为 50 的输出，这个输出将被看作是 0.50。

当 =10001 而<offset>=0 时,输出可以被限定一个最大的精度 0.0001。一个值为 32767 的输入将给出一个值为 5000 的输出，这个输出将被看作是 0.5000。

参见 Analog to Floating Point，Analog Scaler。

27、Numeric Keypad

快速键名：#pad, numpad

信号/参数：10 个数字输入：<0> 到 <9>

3 个数字输入：<clear>, <+> 和 <->

1 个可选数字输入：<enter>

1 个模拟输出：<aout>

1 个可选数字输出：<entered>

1 个参数：<upper limit>（参见 Numeric Formats）

1 个可选参数：<lower limit>

描述：Numeric Keypad 信号在它的输出上产生了由它的数字输入表示的模拟值，并使此值保持在由<upper limit> 和可选参数<lower limit>限定的范围内。

输入<clear>将输出的值初始化为<lower limit>的值，而当<lower limit>没有被定义时，初始化为 0。

输出在<+>的每一个上升沿都增加 1，在<->的每一个上升沿都减 1。

如果输出值达到了<upper limit>，它的值变为 0（或<lower limit>）；如果输出值达到了-1（或<lower limit>减 1），它的值变为<upper limit>。

注意：限定于<upper limit>内的值远多于输出实际上达到的。例如，如果<lower limit>=5 而<upper limit>=1000，输出可假定值为 5 到 99 之间。

可选数字输出信号<entered>在<clear> 或<enter>升高时也升高，并预示下一个数字输入<0>到<9> 由一个新数的第一个数字开始计算。当开始输入时，<entered>变低。

注意：如果输入<enter>已被定义，输出<entered>就必须被定义以确保正确的操作。

二 、CONDITIONAL

1、Analog Compare

快速键名：

compare，acomp

信号

- 两个模拟信号输入：<+> and <->
- 一个可选的模拟信号输入：<->
- 一个数字信号输出：<out>
- 一个可选的数字信号输出：<out*>（即信号 <out>的取反）

描述

在两个输入信号的情况下，Analog Compare 元素比较输入的模拟信号<+>和<->，若输入信号<+>≥<->,则输入高电平;否则输出低电平。

在三个输入信号的情况下，模拟输入信号> and <-> 定义了一个以 <->为上极限和 <-> 为下极限的范围。若输入信号<+>≥<->,则输出为高电平,且保持到<+><<->时为止；若输入信号<+><<->,则输出为高电平,且保持到<+>≥<->时为止。

数字信号输出<out>和可选的数字信号输出<out*>在产生以前是不会产生突变的，也就是说当<out>的状态改变时，<out>和<out*>在瞬间会产生一个相同的值。

2、AND

快速键名

&

信号

- 任意数目的数字输入信号：<i1> 到 <iN>
- 一个数字输出信号：<out>

描述

当所有的数字输入信号<i1> 到 <iN>同时都为高电平，那么 AND 元素输出一高电平的数字信号<out>，否则输出一个低电平的数字信号<out>。

AND 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示

<i1>	<i2>	<out>
0	0	0
0	1	0
1	0	0
1	1	1

3、Binary Decoder

快速键名

decode

信号

- 一个数字输入信号：<enable>
- 任意多的数字输入信号：<i1> t 到 1 <iM>
- 任意多的数字输出信号：<o1> 到 <o*N*>，且 $N = 2^M$

描述

二进字数表示的输入通过 Binary Decoder 元素 转换为一个用一元表示的输出。每个数字输入信号代表了一位二进字数，低电平为 0，高电平为 1。

在两位数的情况下,如果<i1>为低电平并且<i2> 也为低电平(用 0 表示)，那么第一个输出信号<o1> 为高电平。 如果<i1>为低电平<i2>为高电平(用 1 表示),那么<o2>将为高电平。如果<i1>为高电平<i2>低电平(用 2 表示),那么<o3> 将输出高电平,等四种情况。

因此，当有 M 个输入时，将会有 2^M 输出。

当<enable>为高电平时 Binary Decoder 元素在同一时刻只能有一个输出信号为高电平，如果<enable>为低电平时所有输出信号将保持为低电平。

Binary Decoder 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示，无关值用 X 表示：

<enab0e>	<i1>	<i>	<o1>	<o2>	<o3>	<o4>
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

4、Buffer

快速键名

buf

信号

- 一个数字输入信号：<enable>
- 任意多的数字输入信号：<i1> 到 <i*N*>
- 对应每一个输入信号的数字信号输出：<o1> 到 <o*N*>

描述

只要输入信号<enable>和输入信号<i*N*>为高电平，那么和它相对应的输出信号<o*N*>也为高电平。每一个输入对应有一个输出，每个输入/输出对之间是相对独立的。因此 Buffer 信号有时被认为是一个复合 AND 信号，每一个输入信号与<enable>相“与〈AND〉”来决定其相对应的输出信号的。

尽管一个数字信号一般只能有一个驱动源，但是 Buffer 的输出信号是一个例外。也就是说这个输出信号由一个 Buffer 或一个被按下的 Button(或其它系统输入)所驱动的 Buffer 来决定。因此一个独立的 Buffer 能触发多个事件，而多个 Buffer 能使同一组 button 控制不同的设备。在一些应用中，Interlock 元素的输出信号通常被用在 Buffer 的输入信号<enable>上。InterOock 将确定在同一时刻将只能有一个 Buffer 被激活。

Buffer 元素的输出仅仅与其它 Buffer 的输出有关联，而与其它逻辑元素的输出无关。不同系统的输入，例如触摸屏或手持发射器上的按键，也能决定 Buffer 的输出。当多个输出相同时，Buffer 元素的功能将和 Transition Gate 元素相同。

5、Exclusive NOR

快速键名

xnor

信号

- 任意多的数字输入：<i1> 到 <iN>
- 一个数字输出：<out>

描述

在有两个输入的情况下，如果 Exclusive NOR 元素的两个输入信号相同（高电平或低电平）那么将输出一个高电平信号，否则输出低电平 。Exclusive Nor 元素的输出和 Exclusive OR 元素的输出相反。

在多于两个输入的情况下，如果 Exclusive NOR 有偶数个输入信号为高电平或所有输入信号同时为低电平，那么它将输出一个高电平信号，否则将输出低电平信号。

Exclusive NOR 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示。

<i1>	<i2>	<out>
0	0	1
0	1	0
1	0	0
1	1	1

>			>
0	0	0	1
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

6、Exclusive OR

快速键名

xor

信号

- 任何多的数字信号输入：<i1> 到 <iN>
- 一个数字信号输入：<out>

描述

如果 Exclusive OR 元素有奇数个输入信号为高电平，则输出信号为高电平，否则输出信号为低电平。（包括所有输入均为低时的情况）

Exclusive OR 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示。

<i1>	<i2>	<out>
0	0	0
0	1	1
1	0	1
1	1	0

<i1>	<i2>	<i3	<out>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	1

7、NAND

信号

- 任意多的数字输入信号：<i1> 到 <iN>
- 一个数字输出信号：<out>

描述

如果 NAND（negated AND）元素的输入信号中任意一个或多于一个是低电平那么它将输出一个高电平信号，如果所有输入信号都为高电平，那么输出为低电平信号。只有 1 个输入的 MAND 元素相当于 NOR 元素，即输出为输入的反。

NAND 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示。

<i1>	<i2>	<out>
0	0	1
0	1	1
1	0	1
1	1	0

8、Negative Transition Gate

快速键名

ntrans

信号

- 任意多的数字输入信号：<i1> 到 <iN>
- 一个数字输出信号：<out>

描述

Negative Transition Gate 元素输出的信号是它自己最后一个改变的输入信号改变的反转，如果 Negative Transition Gate 元素的输入信号的状态从低电平改变到高电平，将输出一个低电平信号并保持这种状态。

9、NOR

信号

- 任意多的数字输入信号：<i1> 到 <iN>
- 一个数字输出信号：<out>

描述

当所有的输入信号都同时为低电平时 NOR (negated OR)元素的输出为高电平。

NOR 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示。

<i1>	<i2>	<out>
0	0	1
0	1	0
1	0	0
1	1	0

10、NOT

快速键名

!

信号

- 一个数字输入信号：<in>
- 一个数字输出信号：<out>

描述

NOT 元素简单的反转输入信号的电平。也就是说当输入信号为高电平时，输出信号为低电平且一直保持。

NOT 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示。

<in>	<out>
0	1
1	0

11、OR

快速键名

|

信号

- 任意多的数字输入信号：<i1> 到 <iN>
- 一个数字输出信号：<out>

描述

如果 OR 元素有一个或多于一个的输入信号为高电平，那么就输出一个高电平信号，如果所有的输入信号都为低电平，那么就输出低电平信号。

OR 元素可以用以下的真值表来表示：高电平用 1（真）表示，低电平用 0（假）表示。

<i1>	<i2>	<out>
0	0	0
0	1	1
1	0	1
1	1	1

12、Transition Gate

快速键名

trans

信号

- 任意多的数字输入信号：<i1> 到 <i*N*>
- 一个数字输出信号：<out>

描述

Transition Gate 元素输出的信号是它自己最后一个改变的输入信号改变的电平，即如果 Transition Gate 元素的输入信号的状态从低电平改变到高电平，将输出一个高电平信号并保持

13、Truth Table

快速键名

table, tt

信号/参数 (条件和陈述)

- 任意多的数字输入信号：<i1> 到 <i*M*>
- 任意多的数字输出信号：<o1> 到 <o*N*>
- 对应每个输入信号，都有任意多的被测试条件： 0 (0), 1 (1) 或 X (不确定)
- 对应每个输出信号，都有任意多的陈述结果：0 (0), 1 (1), X (不确定) 或 C (补数)

描述

Truth Table 元素为一组特定的输入产生一个指定的输出。它将搜查输入信号的详细列表，直到找到和实际输入状态匹配的。如果没有发现匹配的条件，输出将保持不变。

简单应用

在一个设置了三个座位的远程视频会议中，每一个座位上都有自己的麦克风和视频摄象机，第四个视频摄象机有一个广角镜头能覆盖整个房间。现在 Truth Table 根据麦克风是否在工作来控制摄象机。如果多于一个麦克风工作那么广角摄象机将被激活。

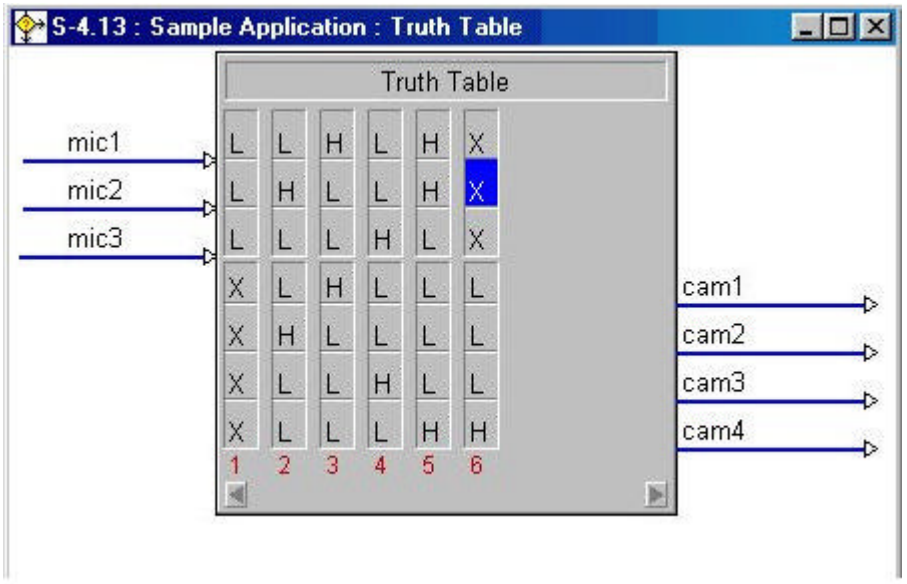
Truth Table 元素将包括以下信号：

- 三个输入：<mic1>，<mic2> 和 <mic3>
- 四个输出：<cam1>，<cam2>，<cam3> 和 <wide_cam>

Input Conditions			*ResuOting Output States*			
<mic1>	<mic2>	<mic3>	<cam1>	<cam2>	<cam3>	<wide_cam>
0	0	0	X	X	X	X
0	1	0	0	1	0	0
1	0	0	1	0	0	0
0	0	1	0	0	1	0
1	1	0	0	0	0	1
X	X	X	0	0	0	1

最后一个条件为任意输入信号的组合，它激活了广角摄象机，考虑到可以用一个简便的方法来指定一个默认的设置，当输入信号与其它输入状态相匹配了，条件的检测就被略过，因此这是合乎约定的。。

图例：



三 COUNTER

1、Binary Counter

快速键名

counter

信号

- 一个数字输入信号：<clock>
- 两个可选的数字输入信号：<reset> 和 <reverse>
- 任意多的数字输出信号：<bit1> 到 <bit*N*>

描述

当<clock>的信号在上升沿时，触发 Binary Counter 按二进制数从 0 开始按升序或降序计数。

例如，在<clock>的第一个上升沿时，<bit1>输出高电平信号（二进制数 1）在<clock>下一个上升沿时，<bit1>输出低电平信号<bit2>输出高电平信号（二进制数 2）。然后<bit1>输出高电平信号并且<bit2>输出也为高电平信号（二进制数 3），等等。

可选项 <reset>输入高电平信号时，将取消计数操作并且所有的输出将为低电平。只要<reset>保持高电平，<clock>的输入将被忽略。（当<reset>变为低电平信号后，将 在<clock>的上升沿从 0 开始计数。可选项<reverse>为高时将变为反序计数。）

2、Ring Counter

快速键名

ring

信号

- 一个数字输入信号：<clock>
- 两个可选的数字输入信号：<reset> 和 <reverse>
- 任意多的数字输出信号：<o1> 到 <o*N*>

描述

当<clock>的信号在上升沿时触发，Ring Counter 按升序或降序计数。在<clock>的每一个上升沿，输出从<o1>开始顺序输出高电平信号。当计数到最后一个输出时，Ring Counter 将返回到<o1>开始下一轮计数。

输出信号是突变的，也就是说在同一时间仅有一个输出为高电平信号。如果在〈clock>的上升沿，可选项<reverse>为高时，将变为反序计数。可选项 <reset>输入高电平信号时，将取消计数操作并使<o1>输出高电平。只要<reset>保持高电平，<clock>的输入将被忽略。

在刚开始没有输出信号为高电平，只要<clock>（或 <reset>引起<o1> 输出高电平信号，Ring Counter 元素以后将不会再次出现这种状态。

四 DEGUGGING

1、Analog Debugger

快速键名

test2, adebug

信号

- 任意多的模拟信号输入：<i 1> 到 <i N>

描述

在程序运行期间，Analog Debugger 元素将跟踪模拟信号，把其输入信号传输到 Viewport 控制台。

在 ST-CP 和 CN 系列的控制主机中,这个元素能把信号传输到 Test Manager 控制台。

2、Message to Computer Port

快速键名

tmsg

信号/参数

- 任意多的数字输入信号：<trig1> 到 <trigN>
- 一个参数：<String>

描述

在任意<trig>输入信号的上升沿 Message To Computer Port 元素把<String>参数中指定的字符串传送到 Viewport 控制台。

3、Serial Binary to Hex

快速键名

bin>hex, b/h, btoh

信号

- 一个字符串输入信号：<bin$>
- 一个字符串输出信号：<hex$>

描述

Serial Binary To Hex 元素将把输入的二进制字数符串转换为十六进制数字符串；转换后的字符将通过一个 Serial Debugger (ASCII) 元素传输到 Viewport 控制台。

4、Serial Debugger (ASCII)

快速键名

test1, sdebuga

信号

- 任意多的字符串输入信号：<i1> 到 <i*N*>

描述

Serial Debugger (ASCII) 元素把信号以 ASCII 形式传输到 Viewport 控制台。

当输入信号不是 ASCII 格式，Serial Binary to Hex 元素可将数据由二进制数转换成十六进制数。转换后的字符将通过一个 Serial Debugger (ASCII) 元素传输到 Viewport 控制台。但采用 Serial Debugger(Hex)格式更好，因为它可以接受任何格信号的输入的。

5、Serial Debugger (Hex)

快速键名

test0, sdebugh

信号

- 任意多的字符串输入信号：<i1> 到 <i*N*>

描述

Serial Debugger (Hex)元素可以接受任何格式的字符串，然后已十六进制格式输出到 Viewport 控制台。

五 MEMORY

1、Analog Non-Volatile Ramp

快速键名

rampnv

信号/参数

- 两个数字输入信号：<up> 和 <down>
- 一个可选数字输入信号：<mute>
- 一个模拟输入信号：<aout>
- 一个双精度数参数：<ramp_time>

描述

当输入信号<up>或<down>为高电平时，Analog Non-Volatile Ramp 元素产生一个线性改变的模拟信号输出，然将结果储存到 NVRAM 中。

<ramp_time>参数指定了输出信号从 0% 上升到 100% 所需要的时间。在开始时输出的信号的值是前一个参数时间<ramp_time>所储存的值。

在可选输入信号<mute>的上升沿，输出信号为 0%,且在下降沿返回先前的值。 如果要保持输出信号为 0%，需通过 Toggle 元素来驱动输入信号<mute>。

当输入信号<up> 和 <down>为高电平，将不考虑输入信号<mute>的状态，并开始下一个 Ramp 操作。因此在输入信号<mute>为高电平时建议用 Buffer 元素来禁止<up> 和 <down>的输入。

2、Analog RAM

快速键名

ram，aram

信号

- 两个数字输入信号：<store> 和 <recall>
- 任意多的模拟输入信号：<ain1> 到 <ainM>
- 任意多的数字输入信号：<select1> 到 <selectN>
- 对应每一个模拟输入信号的模拟输出信号：<aout1> 到 <aoutM>

描述

当任意一个或多个输入信号<select>和输入信号<store>同时输入高电平信号时 Analog RAM 元素将把所有模拟输入信号<ain1> 到 <ainM>的当前值储存到 NVRAM 中，当任意一个输入信号<select>和输入信号<recall>同时输入高电平信号时 Analog RAM 元素将把这组储存的值通过<aout1> 到 <aoutM>输出（每一个储存的值都和一个输入信号<select>对应)。每一个模拟输入信号都对应一个模拟信号输出，并且每一组输入/输出相对独立。(通常每一个输入信号和输出信号的名称相同。)

当输入信号<recall>为高电平时，同时只能有一个输入信号<select>为高电平（也就是说，输入信号<select>通常要用 INTERLOCK 元素进行互锁）。这意味着同时只能有一个设置值被呼出。为了能在同一时间呼叫多组设置值，必须用 Analog RAM from Database 元素，它的定义和 Analog RAM 相同。

当<store> 或 <recall>是脉冲信号或反转时输入信号<select>将被锁在高电平,同样当<select>是脉冲信号时，输入信号<store> 或 <recall>将被锁在高电平。

假设输入信号<recall>的信号名称是“1”，在启动时，哪一个输入信号<select>为高电平，输出信号就被设置为和它相对应的储存值。

如果输入信号<store> 和 <recall> 的信号名都为“1” 并且每一个模拟输入信号和输出信号名相同，那么信号将处于不停的循环中。在启动时，哪一个输入信号<select>为高电平，输出信号就被设置为和它相对应的储存值。

当 Analog RAM 元素 只有一个输入信号时，输入信号<select> 的信号名称可以是“1”。

3、Analog RAM from Database

快速键名

ram2，ramdb，aramdb

信号

- 两个数字输入信号：<store> 和 <recall>
- 任意多的模拟输入信号：<ain1> 到 <ain*M*>
- 任意多的数字输入信号：<select1> 到 <select*N*>
- 对应每一个模拟输入信号的模拟输出信号：<aout1> 到 <aout*M*>

描述

Analog RAM 元素指向先前定义的 Analog RAM 元素所指向的内存。因为 Analog RAM 不能同时呼出多个预设值（当输入信号<recall>为高电平时，输入信号<select>同时只能有一个为高电平信号）,Analog RAM from Database 元素能用来呼出第二个预设值。用来指向单一 Analog RAM 元素的符号个数并无限制。

Analog RAM from Database 元素的参数必须和 Analog RAM 元素定义的一样（例如：数目，输入、输出的信号名）。Analog RAM 元素必须先于 Analog RAM from Database 定义。

4、D Flip Flop

快速键名

dff

信号

- 两个数字输入信号：<clock> 和 <data>
- 两个可选数字输入信号：<set> 和 <reset>
- 一个数字输出信号：<out>
- 一个可选的数字输出信号：<out*> (<out>的补数)

描述

当输入信号<clock>处于上升沿时，D Flip Flop 元素的输出信号<out>将和输入信号<data>相匹配，输出信号<out*>和<out>相反。

当可选输入信号<set>为高电平时,输出信号<out>为高电平;当输入信号<reset>为高电平时,元素的输出信号<out>为低电平。

输出信号<out>和<out*>是不可突变的，这意味着当<out>的状态改变时<out>和<out*>将在瞬间有相同的值。

5、Digital RAM

快速键名

dram

信号

- 两个数字输入信号：<store> 和 <recall>
- 任意多的数字输入信号：<i1> 到 <i*M*>
- 任意多的数字输入信号：<select1> 到 <select*N*>
- 对应每一个输入<i1> 到 <i*M*>，都对应一个数字输出信号：<o1> 到<o*M*>

描述

Digital RAM 与 Analog RAM 香对应.当任意一个或多个输入信号<select>和输入信号<store>同时为高电平时，Digital RAM 元素将当前所有输入<i>的值储存到 NVRAM 中,并且当任意一个或多个输入信号<select>和输入信号<recall>同时为高电平时，Digital RAM 元素将设置输出信号为当前储存在 NVRAM 中的值。（每一个存储在内存中的值都和一个输入<select>相对应。） 每一个输入<i>都和一个输出<o>相对应,且每组输入/输出相对独立的。

当输入信号<recall>为高电平时，输入信号<select>同时只能有一个为高电平信号（也就是说，输入信号<select>通常要用 INTERLOCK 元素进行互锁）。这意味着同时只能有一个设置值被呼出。

当输入信号<store> 或 <recall>的输入是脉冲信号或反转时输入信号<select>将被锁在高电平，同样当输入信号<select>是脉冲信号时，输入信号<store> 或 <recall>将被锁在高电平。

如果输入信号<recall>的信号名称是“1”，在启动时，哪一个输入信号<select>为高电平，输出信号将被设置为和它相对应的储存值。

如果输入信号<store> 和 <recall> 的信号名都为“1”并且每一组输入/输出有相同的信号名，那么信号将处于不停的循环中。在启动时，哪一个输入信号<select>为高电平，输出信号将被设置为和它相对应的储存值。

当 Digital RAM 元素 只有一个输入信号时，输入信号<select> 的信号名称可以是“1”。

6、FIFO Queue

快速键名

que

信号

- 三个数字输入信号：<enable>，<clear> 和 <pop>
- 任意多的数字数字输入信号：<select1> 到 <select*M*>
- 两个模拟输出信号：<top> and <#qued>
- 任意多数的连续输出信号：<display-1$> 到 <display-*N*$>
- 对应每一个数字输入信号的数字输出信号：<select1-fb> 到 <select*M*-fb>

每一个数字输入信号<select1> 到 <select*M*> 都必须有一个相对应的数字输出信号 <select1-fb> 到 <select*M*-fb>。 FIFO Queue 元素不能自动的扩展共同的部分，只能手动扩展。

描述

先进先出(FIFO)的队列是一种比较常用的数据结构。这就是说，按它们插入的顺序取出。

在 SIMPL 中,FIFO Queue 元素允许先定义的 Serial RAM 元素被插入并且从队列中移出。模拟输出信号<#qued>给出了当前队列中元素的数量,模拟输出信号 <top>给出了下一个移出的元素距离队列顶端元素的位置（从 1 到 M）。

数字输入信号<pop>用于获取模拟输出信号<top>,之后就将其从程序队列中移走,并将所有的元素升高。当数字输入信号<clear>为高电平时，队列被清空。

输入信号<select>必须和相对应的 Serial RAM 元素输入信号 <select> 的名字相同。当输入信号<enable>为高电平并且输入信号<select>为高时,元素能被增加到队列中。当一个元素被增加时，输出信号<select-fb>为高电平。当这个元素从队列中移出<select-fb>输出低电平信号。(当输入信号<pop> 或 <clear>为高电平时，输入信号<enable>无效。)

连续输出信号<display$>用来显示目的。例如，当一个 BUTTON 按下时从队列中增加或移出的元素可以通过间接文本显示在触摸屏上。输出信号<display$>的数量不一定不等于输入信号<select>的数量。如果队列中包含的元素比定义的输出信号<display$>多，字符串输出信号将显示在队列中最顶的几个，并且在元素移出时更新。

多个 FIFO Queue 元素能指向一个 Serial Ram 元素；多个 Ram/FIFO Queu 对能被定义在一个程序中。

7、Interlock

快速键名

ilock, il

信号

- 两个可选的数字输入信号：<clear> 和 <set all>
- 任意多的数字输入信号：<i1> 到 <i*N*>
- 对应每一个数字输入的数字输出信号：<o1> 到 <o*N*>

描述

在输入信号的上升沿，Interlock 元素输出一个高电平信号并且锁住，并使其他输出信号为低电平。每一个输入信号都有一个对应的输出信号，每一组输入/输出相对独立的。

Interlock 元素记忆了最后输入高电平的状态，因而不管在输入如何改变输出将保持高电平。另外，所有的输出是能突变的，这意味着前一个被激活的输出在下一个输出为高电平前变为低电平。这个逻辑在许多应用程序是非常便利的，特别是当互锁用于多个 Buffer 元素的<enable> 输入时。（突变的特性确保在同一时间内只有 1 个 Buffer 可以被激活）

可选输入信号<clear>为高电平时将使所有的输出为低电平。可选输入信号<set all> 将使所有的输出同时为高电平（这在所有的输出都能同时为高电平时）。这对某些实际应用是非常有用的，例如用 Analog，Digital，或 Serial RAM 元素初始化内存。

8、JK Flip Flop

快速键名

jk

信号

- 两个可选的数字输入信号：<set> 和 <reset>
- 三个数字输出信号：<clock>，<J> 和 <K>
- 一个数字输出信号：<out>
- 一个可选的数字输出信号：<out*> (和<out>相反)

描述

JK Flip Flop 元素根据输入信号<J>和<K>的状态，在输入信号<clock>的每一个上升沿使得输出信号为高电平或低电平，并使之保持：

- 如果 <J>和<K> 都是低电平,输出保持不变。
- 如果 <J>和<K> 都是高电平，输出电平就套牢（toggles）或反转。
- 如果 <J>是高电平而<K>是低，输出改变或升高。
- 如果 <J>是低电平而<K>是高电平，输出复位或降低。

可选输入<set>和<reset> 同时对 Set/ Reset Latch 元素的输入<set>和 <reset>进行作用 ；<set>迫使输出升高，<reset>迫使输出降低

注意：<out>和可选信号<out*>不会突变。这表明当<out>的状态改变时，<out>和<out*>都在瞬时有相同值。

9、Memory Interlock

快速键名

mi, milock

信号

- 一个可选的数字输入信号：<clear>
- 任意多的数字输入信号：<func1> 到 <func*M*>
- 任意多的数字输入信号：<src1> 到 <src*N*>
- 对应每一个输入信号<func1> 到 <func*M*>的数字输出信号：<func1-fb> 到 <func*M*-fb>
- 对应每一个输入信号<src1>到<src*N*>的数字输出信号：<src1-fb> 到 <src*N*-fb>

描述

Memory Interlock 元素允许通过 1 个共享的控制设置来控制任意数量的组件，互锁的<func>信号代表共享的控制（如在传输功能中），而互锁的<src>信号代表那些组件（如在数据源中）。

当可选输入信号<clear>为高电平时，元素将被重新初始化，清除所有当前和存储的状态。

应用实例

考虑到有 5 台 VCR，每台都有一些相同的控制．输入信号<func1> 到 <func5> 的信号名为 *Play*，*Stop*，*Mute*，*Rewind* 和 *Fast Forward*， 并且输入信号 <src1> 到 <src5>的信号名为 *VCR-1*，*VCR-2*，等等。

当 VCR-4 被选择(按键被按下)，输入信号<src4> 将激活输入信号<func>，因此最终用户能按下 *Play* 或其它互锁的按键来控制 VCR-4，随之输出信号<func-fb> 将提供适当的互锁反馈信号。然后如果 VCR-2 被选择，Memory Interlock 将记住 VCR-4 的状态，而输入信号<src2>将重新激活输入信号<func> 和输出信号来控制 VCR-2。

10、Serial Memory Search

快速键名

ismem, smsearch

信号/参数

- 一个连续的输入信号：<in$>
- 两个数字输入信号：<trig> 和 <enable>
- 一个可选的数字信号：<gate>
- 任意多数字输出信号：<selected-1> 到 <selected-*N*>
- 一个参数：<#chars>

描述

Serial Memory Search 元素从预先被定义的 Serial RAM (或 Serial RAM from Database) 元素的储存值中搜索和输入信号<in$>相匹配的一个或多个元素。每一个 Serial Memory Search 元素的输出信号<selected-*N*> 对应一个 Serial RAM 元素的输入信号<select*N*>，并且当一个匹配的元素被发现，相应的输出信号<selected> 为高电平。(输出信号<selected> 的数目将小于等于相应的 Serial RAM 元素的输入信号<select>)

当输入信号<in$>为静态字符串时,输入信号<trig> 将被使用。例如使用 phone Dialing Keypad。当输入信号 <enable> 为高电平时，Serial Memory Search 元素在输入信号<trig> 的每一个上升沿开始搜索。

可选输入信号<gate>可用作过渡字符串。只要一个连续信号被更新并且输入信号<enable>和<gate>都为高电平时 Serial Memory Search 元素便开始搜索。(在输入信号的边沿无效。)

当输入信号<enable>为高电平信号将激活输入信号<trig> 和 <gate>。在输入信号<enable>的下降沿,所有的输出信号<selected>输出为低电平信号。

参数<#chars>必须和 Serial RAM 元素中的参数<#chars>相同。

11、Serial Queue

快速键名

sque

信号/参数

- 一个连续输入信号：<in$>
- 两个数字输入信号：<xfer>和<clear>
- 一个连续输出信号：<out$>
- 一个参数：<queue_size>

描述

Serial Queue 元素以字符的形式来储存输入的字符串并且在输入信号<xfer>的上升沿传输整个连续信号。除非数据在队列中出现否则传输不会发生。

输入信号<clear>通常和<xfer>一样有约束作用，在连续信号传输后清空队列。如果这些信号没有一起约束，那么在输入信号<xfer>的每一个上升沿 Serial Queue 将重传 Serial Queue 中的内容。

参数<queue_size>指明了能被增加到队列中的字符数字的形式，比如指定最大输入不能超过 255 个字符。（如果超过了最大数，错误信号将传送到控制台。）

12、Serial RAM

快速键名

smem, sram

信号/参数

- 一个连续输入信号：<in$/out$>
- 两个数字输入信号：<store> 和 <recall>
- 任意多的的数字输入信号：<select1> 到 <select*N*>
- 一个数字输出信号：<dial>
- 一个参数：<#chars>

描述

通常 Serial RAM 元素和 Telephone Dialing Keypad 元素或 ASCII Keypad 元素一起使用把连续信号储存在系统内存（NVRAM）中。也就是说后面两个元素产生的字符串储存和调出。Serial RAM 元素同样能和 Serial Memory Dialer 元素一起使用作为一个自动拨号器。

Serial RAM 元素不能和产生静态数（例如：Analog to Serial 元素或 Serial I/O 元素 ）一起使用，因为静态数不能被存储到系统内存中。

Serial RAM 元素通常它的输入信号<in$/out$>是被用来储存和调出的数据。因此它的输入信号必须和 Serial Memory Dialer 的输入信号<in$> ，它的输出信号必须和 Keypad 元素的输出信号同名。

当任一个或多个输入信号<select>和输入信号<store>同时为高电平时 Serial RAM 元素将把它的输入信号<in$/out$>的值存储到系统内存中，并在一个输入信号<select>和输入信号<recall>同时为高电平时调出数据。(每一个被存在内存中的数的和一个输入信号<select>相对应)

当数据被调出时，输入信号<in$/out$>将把 Keypad 元素的内容设置为储存的值。输入信号<in$/out$>通过连接到一个触摸屏的连续输出的方式，可以将数据发送到触摸屏的直接文本域中。

在大多数应用程序中输入信号<store> 或 <recall> 将被保持为高电平而输入信号<select>作为脉冲信号。

当输入信号<recall>为高电平时，输入信号<select>同时只能有一个为高电平信号。这意味着同时只能有一个设置值被呼出。为了同时呼出多余一个的值需要用 Serial RAM from Database 元素，它的定义参数和 Serial RAM 元素相同。

输出信号<dial>的一个典型应用是输出到 Serial Memory Dialer 元素的输入<dial>上来拨一 个储存的号码。

参数<#chars>指定了能为每一个元素储存的最大数值。

13、Serial RAM from Database

快速键名

smem2，sramdb

信号/参数

- 一个连续输入信号：<in$/out$>
- 两个数字输入信号：<store> 和 <recall>
- 任意多的的数字输入信号：<select1> 到 <select*N*>

- 一个数字输出信号：<dial>
- 一个参数：<#chars>

描述

Serial RAM From Database 元素指向先前定义的 Serial RAM 元素所指向的内存。因为 Serial RAM 不能同时呼出多个预设值(当输入信号<recall>为高电平时,输入信号<select>同时只能有一个为高电平信号），Serial RAM from Database 元素能用来呼出第二个预设值。

Serial RAM from Database 元素的参数必须和 Serial RAM 元素定义的一样（例如：数目，输入、输出的信号名）。Analog RAM 元素必须先于 Analog RAM from Database 定义。

14、Set/Reset Latch

快速键名

sr

信号

- 两个数字输入信号：<set> 和 <reset>
- 一个数字输出信号：<out>
- 一个可选的数字输出信号：<out*>（和输出信号<out>相反）

描述

在输入信号<set>的上升沿，Set/Reset Latch 元素锁住它的输出为高电平信号，在输入信号<reset>的上升沿将锁住它的输出为底电平信号。

在大多数应用程序中，它被建议用于将<set>或<reset>输入设为脉冲信号而不是用于将它们锁定在高电平或低电平。当这些输入被锁定时，整个元素就在每一次输入改变的时候被重新求值，这将导致不可预知的后果。

输出信号<out>和可选的输出信号<out*>是不能突变的,这意味着当输出信号<out>的状态改变时，输出信号<out>和<out*>将在改变的瞬间有相同的值。

15、Toggle

快速键名

toggle

信号

- 一个数字输入信号：<clock>
- 两个可选的数字输入信号：<set> 和 <reset>
- 一个数字输出信号：<out>
- 一个可选的数字输出信号：<out*>（和<out>相反）

描述

在输入信号<clock>的每一个上升沿 Toggle symbol 将锁定输出信号为高电平或低电平在输入信号<clock>的每一个上升沿。

可选输入信号 <set>和<reset>的作用和 Set/Reset Latch 元素中的输入信号<set>和<reset>相同，当输入信号<set>为高电平时 强迫输出为高电平信号，当输入信号<reset>为高电平时强迫输出为低电平。

输出信号<out>和可选的输出信号<out*>是不能突变的，这意味着当输出信号<out>的状态改变时，输出信号<out>和<out*>将在改变的瞬间有相同的值。

六、Serial

1、Analog to Serial

快速键名：txa, a/s, atos

信号/参数：1 个数字输入：<**trig**>

任意数量的模拟输入：<**ain1**>**到**<**ain*M***>

1 个连续输入：<**out$**>

任意数值的参数：<**String1**>**到**<**String*N***>，N **值从** 1 **到** M+1

1 **个参数：**<**Format**>

注意：参数<**String**>**的列表可以用** Alt+**命令** **的方式被扩展。**

描述：Analog to Serial **元素从它的模拟输入的低字节（或高字节）或** 1 **个联合模拟输入和字符串常量中构造了** 1 **个字符串，然后在**<**trig**>**的每一个上升沿传送这个字符串。用这种方式，元素能够更新和传送在运行时间内改变了的数据。**

<**Format**>**参数决定是否要从模拟输入的低字节或高字节里接受数据。它也将那些特殊的格式列入清单，即使这些格式在被传送之前是应用于字符串的。关于**<**Format**>**将在后面的部分作详细的描述。**

输出字符串是这样构造的（假设设置为低字节）：

如果没有用到字符串常量，每一个模拟输入的低字节就连接起来（从<**ain1**>**开始顺序进行）然后相应的字符串被传送。比如，如果从**<**ain1**>**到**<**ain5**>**的低字节为以下几个十六进制值：**

48h, 65h, 6Ch, 6Ch and 6Fh

字符串“Hello**”就被传送。在大多数情况下，**Analog Initialize **元素将用于给模拟输入赋值。（此值可以用可选择的字符形式来传送：**'H', 'e', 'l', 'l', 'o'**。）就像前面规定的，字符串被更新并在**<**trig**>**的每一个上升沿被传送。**

有时应用程序要求字符串常量内含在输出数据中。这里，元素用顺序连接<**String1**> + <**ain1**> + <**String2**> + <**ain2**> + <**String3**> + <**ain3**>+---- **的方式构造了输出字符串。任何没有用到的参数都要被定义并被赋值为空字符串（“ ”）。例如，假设输出字符串将由以下信号和参数构成：**

Two analog signals (<**ain1**> and <**ain2**>)

The string "TEST" (<**String3**>)

Two analog signals (<**ain3**> and <**ain4**>)

The string "\x0D\x0E" (<**String5**>)

在上面这个例子中，<**String1**>, <**String2**>**和**<**String4**>**必须被定义并赋值为空字符串（“ ”）。（如果不需要插入字符串，将不需要填写所有**<**String**>**参数。）**

格式： **如上所述，**<**Format**>**参数适用于以下两种不同的情况：①指定每一个模拟输入要减小的字节；②指定特殊的格式，即使此格式是应用于传送前的字符串的。作为一个 16 位数，**<**Format**>**可以在逻辑上自己分成一个低字节和高字节，每一个字节用于一个不同的用途，见下面的叙述。**

要指定用模拟输入的哪一个低字节或高字节，调整<**Format**>**的第 9 位的值。如果此位为 0，就用高字节（不推荐用于大多数应用程序中）。要用低字节（推荐），可通过在**<**Format**>**的总值上加十进制数 256 的方法来设置第 9 位为 1。**

参数的低字节可以用于添加一个求校验和（checksum**）的特殊类型到输出字符串上，如下表。**

<Format>(low byte)	Suffix
0	No formatting information.
1	A longitudinal parity byte, which is the XOR of all other bytes in the string.
2	A binary checksum (add all the bytes in the string and use the lower byte of the result).
3	A twos complement checksum (add all the bytes in the string and take the twos complement of the lower byte).
4	A two-byte ASCII checksum followed by a line-feed character (used by Grass Valley Video Switchers).
5	A ones complement checksum (add all the bytes in the string and take the ones complement of the lower byte.
6	A Barco monitor checksum.
7	An extended Barco monitor checksum.
8	Format 2, the high bit (bit 7) set to 0.
9	A checksum used for Somfy motor controllers.
10	A checksum used for Unity HVAC systems.
11	A Grass Valley 7000 checksum.
12	HAI protocol header and CRC.
13	EIB protocol.

格式举例：用高字节并加一个二进制的校验和：<**Format**>=2
用低字节并加一个补码的校验和：<**Format**>=259 (256 + 3)
用未指定格式的低字节：<**Format**>=256**（最普遍的用法）**
参见 Analog Initialize。

2、ASCII Keypad

快速键名：sdac2, asciipad
信号/参数：3 个数字输入：<**clear**>, <**enable**> **和** <**backspace**>
最多 127 个数字输入： <**HEX-00**>**到**<**HEX-7E**>**（用标准** ASCII **码表示）**
1 个连续输入：<**out$**>
任意数量的参数：<**String1**>**到**<**String*N*>**，**N **值从** 1 **到** M+1
1 **个参数：**<**#chars**> (*See* Numeric Formats)
描述：只要<**enable**>**为高电平，**ASCII Keypad **元素就从它的数字输入中产生一个静态（**static**）的字符串。**
这样，它就和 Telephone Dialing Keypad **元素的操作相同，除了** Telephone Dialing Keypad **元素的输入表现为数字而** ASCII Keypad **元素可以表现为任意标准** ASCII **码。**
注意：输出字符串的静态特性允许它通过 Serial RAM **元素储存在内存（**NVRAM**）中。**
启动时，ASCII Keypad **元素设立一个内部的缓冲器，这个缓冲器的大小就是**<**#chars**>**参数的大小。**

（<#chars>参数与 Telephone Dialing Keypad 元素的<#chars>参数相同）如果加入的字符超过了在<#chars>参数中指定的数量，超额的字符将被忽略。

由于每一个数字输入都升高了，相应的 ASCII 码堆积在缓冲器中，整个字符串将被重新传送。例如，如果输入构成了字符串"TEST"，此元素将先传送"T"，然后是"TE"，接着是"TES"，最后为"TEST"。

<clear>输入移动当前积累的字符串并传送 1 个空字符串。<backspace>输入就像回退键（backspace）一样操作，使最终用户无须知道全部序列就能够编辑接口程序。

当<enable>为低电平时，所有的输入都被忽略（包括<clear>和<backspace>）。当<enable>变为低电平时，如果有积累的字符串，则字符串在指针中将保持，并在<enable>为高时被重新传送。

参见 Telephone Dialing Keypad, Serial RAM, Serial Memory Search

3、ASCII Serial Decoder

快速键名：srcl, asciidecode

信号/参数：1 个连续输入：<in$>

1 个数字输入：<dial>

1 个数字输出：<busy>

最多 127 个数字输出：<HEX-00>到<HEX-7E>（表现为标准 ASCII 字符）

1 个双精度参数：<time_betwn_chars> (参见 Numeric Formats)

描述：ASCII Serial Decoder 元素的操作和 Serial Memory Dialer 元素相同，除了 Serial Memory Dialer 元素的输出表现为数字而 ASCII Serial Decoder 元素的输出可以表现为任何标准 ASCII 字符。

ASCII Serial Decoder 元素通常与 ASCII Keypad 元素和 Serial RAM 元素一起工作，如下：

ASCII Keypad 元素产生 1 个静态字符串。它的输出与 Serial RAM 元素（此元素将字符串存储在内存里）和 ASCII Serial Decoder 元素的<in$>输入相关。

通常，Serial RAM 元素的<dial>输出驱动 ASCII Serial Decoder 元素的<dial>输入。当<dial>升高时，<in$>从内存中唤醒，相应的数字输出顺序升高。当出现"dialed"字符串时，<busy>输出升高。

<time_betwn_chars>参数控制拨号操作的速度并指定从一个输出降低到下一个输出升高之间的时间。在同一时间内只有一个输出为高电平。

参见 Serial RAM, Serial Memory Dialer, ASCII Keypad, Telephone Dialing Keypad

4、Duple Encoder

快速键名： duple

信号/参数：1 个连续输入：<in$>

1 个连续输出：<out$>

描述：Duple Encoder 元素和 Duple Decoder 元素的典型应用是用于使连续数据可以在 Crestron 控制系统和 Unity HVAC 单位之间来回传递。Duple Encoder 元素将连续数据转换成 Unity 系统要求的格式，而 Duple Decoder 元素将数据转换回标准格式。

在双重编码设计中，输入字符串的每一个字节被分成两个“半字节”(4 位)，分组叫做“半位元组”（或 *nybbles*）。例如，字符"H"（ASCII 码 48，十六进制）的上半位元组为 4 而下半位元组为 8。

每一个半位元组像一个字节一样被编码，如下表所示：

4-Bit Value (Hex)	Translated Byte (Hex)

0	5A
1	59
2	6A
3	69
4	56
5	55
6	66
7	65
8	9A
9	99
A	AA
B	A9
C	96
D	95
E	A6
F	A5

若输入字符串"HELLO"**：**

\x48\x45\x4C\x4C\x4F

双重编码输出变成：

\x56\x9A\x56\x55\x56\x96\x56\x96\x56\xA5

当第一个字符“H**”(**ASCII **码** 48**，十六进制）变成**\x56\9A **时，第二个字符“**E**”(**ASCII **码** 45**，十六进制）就变成**\x56\x55**，依次类推。**

参见 Duple Decoder, Analog to Serial

5、Duple Decoder

快键名称：unduple

信号：1 **个连续输入：** <in$>

1 **个连续输出：** <out$>

描述：Duple Decoder **元素和** Duple Encoder **元素的典型应用是用于使连续数据可以在** Crestron **控制系统和** Unity HVAC **单位之间来回传递。**Duple Encoder **元素将连续数据转换成** Unity **系统要求的格式，而** Duple Decoder **元素将数据转换回标准格式。**

注意：Duple Decoder **元素不支持处理一个完整的信息包。因此，要确保一个完整的信息包能够传递到** Duple Decoder **元素，就要用到** Serial Gather **元素及** C9h **的<**delimiter**>参数。这是一个** Unity HVAC **系统里的** ETX **（**end of text**）字符。（通常，**Serial Gather **元素的<**length**>参数必须至少和最长的信息包一样长。）**Duple

Decoder 元素检查是否有有效的 HVAC STX (start of text)字符（0xB4）以及最后字符（0xC9，就如描述的一样）；如果缺少这些字节，Duple Decoder 元素将不发出输出。所以，当信息包被解码时，最后两个字节（解码后的校验和）将不传给输出。

参见 Duple Encoder, Analog to Floating Point, Floating Point to Analog, Serial Gather

6、Serial/Analog One-Shot

快键名称：smv, s1shot, a1shot, sos, aos

信号/参数：1 个连续或模拟输入：<**rx$**>

1 个数字输出：<**out**>

1 个可选数字输出：<**out***>（<**out**>的组件）

1 个双精度参数：<**pulse_time**> (参见 Numeric Formats)

描述：Serial/Analog One-Shot 元素在由<**pulse_time**>指定的期间内驱动它的输出信号在输入信号改变时升高，在<**pulse_time**>失效后驱动它的输出信号降低。

Serial/Analog One-Shot 元素可以被重触发，意思是说它可以辨识输入的任何改变，甚至当<**out**>为高电平而导致它重新开始计算时也是如此。直到全部<**pulse_time**>已不间断流逝，输出信号才降低。

注意：<**out**>和可选信号<**out***>不是突变信号。这意味着当<**out**>的状态改变时，<**out**>和<**out***>都将在同一瞬时拥有相同的值。

7、Serial Buffer

快键名称：sbuffer, sbuf

信号/参数：1 个数字输入：<**enable**>

任意数量的连续输出：<**in1$**> 到 <**in*N*$**>

每一个连续输入都有一个对应的连续输出：<**out1$**>到<**out*N*$**>

描述：只要<**enable**>为高电平，Serial Buffer 元素就传送它的连续输入到它们相应的输出上。每一个输入都有一对相应的输出，每一个输入/输出对之间是相互独立的。

同其他的 Buffer 元素一样，Serial Buffer 元素的输出可以被“压”在一起。不过，如果多重输入同时传播到“压”在一起的输出时，只有 1 个字符串可以被传送——其他的将丢失。因此，如果应用程序需要发出所有的字符串时，通过 Serial Concatenation 元素来路由输出信号就是非常必要的。

注意：与 Analog Buffer 元素不同，Serial Buffer 元素在<**enable**>的上升沿或下降沿不传播信号，而是允许信号在<**enable**>为高电平时不停的流动。这是十分必要的，因为大多数连续信号是短暂的。

比较 Analog Buffer, Serial Concatenation 元素

8、Serial Concatenation

快键名称：sum$, cat$

信号/参数：任意数量的连续输入：<**in1$**> 到 <**in*N*$**>

1 个连续输出：<**out$**>

描述：Serial Concatenation 元素可以同时接收多重连续输入信号。然后在连续逻辑波上每次传送一个字符串，从<**in1$**>开始，依次进行。

用这种方式，Serial Concatenation 元素不需逐字的连接字符串，而是交错进行传送来确保不丢失数据。比较典型的是，Serial Buffer 元素的压入的输出由 Serial Concatenation 元素从彼此覆盖状态路由到预设的字符串上。

参见 Serial Buffer

9、Serial Demultiplexor

快键名称：sdemux

信号/参数：1 个连续输入：<**in$**>

1 个可选模拟输入：<**selector**>

任意数量的连续输出：<**out1$**> 到 <**out*N*$**>

1 个参数：<**form**>

描述：*multiplexor* 使复合信号能够沿着单一的路径被传送，而不丢失任何单独信号的整数部分。原始信号在接收端由 *demultiplexor* 重组。

Serial Demultiplexor 元素以如下两种方式传播它的输入到一个特定的输出上：

当<**form**>参数为 0 时，字符串被路由到与<**selector**>输入值匹配的输出上。例如，若<**selector**>的值为 7，字符串就被路由到<**out7$**>输出。若<**selector**>的值为 0，将不传播任何数据。

当<**form**>=1 时，输入的字符串必须加上前缀：

\x1B\x*MM*

这里 *MM* 表示字符串将被路由到的输出信号。*MM* 从 21 开始，即指定<**out1$**>输出（<**form**> = \x1B\x21）。这样，要路由字符串到<**out7$**>输出，字符串必须加上前缀\x1B\x27。如果 *MM*=20，则不传播任何数据。要把\x1B 当作输出数据的一部分来传送，就要用\x1B\x1B，这里只有一个\x1B 可以通过输出。

注意：<**form**>参数只有两个有效值：0 或 1。

10、Serial Demultiplexor (Special)

快键名称：smtx

信号/参数：1 个连续输入：<**rx$**>

任意数量的连续输出：<**tx1$**> 到 <**tx*N*$**>

1 个参数：<**hdr/list**>

描述：*multiplexor* 使复合信号能够沿着单一的路径被传送，而不丢失任何单独信号的整数部分。原始信号在接收端由 *demultiplexor* 重组。

在 SIMPL 里，Serial Multiplexor (Special)元素通过一个单一的输出路径来传送复合字符串，而 Serial Demultiplexor (Special)元素重组原始字符串。

注意：既然这两个元素是相互对称的，那么这两个元素的参数也必然是严密吻合的。

<**hdr/list**>参数可以用两种方法来区别字符串：一种是为每一个字符串指定一个头；另一种是确定每一个字符串的位置（从 1 到 N）。Multiplexor 把这两条信息加到每一个字符串上，而 Demultiplexor 利用这个信息将字符串分开并路由它们到相应的输出上。

"hdr"部分指定了一个 2 字节的头，这个头可以是以下四个字符中的任意一个：4*, --, AB 或 \x45\x33。"list"部分由每一个字符串中的一个字符组成。也就是说，给出五个字符串，（<**tx1$**> 到 <**tx5$**>），"list"部分应该是由任意五个字符组成，如：ABCDE, qwert, 或 \x43\x23\x9A\x21\x33 等等。

因此，拥有五个字符串的<hdr/list>参数看起来应该如下所示：

4*ABCDE
--qwert
AB\x43\x23\x9A\x21\x33

以上面的第一个例子来说，如果字符串 "TEST" 得到值为<tx2$>，Multiplexor将在字符串前加头"*4"和字符"B"作为前缀，字符是对应于<tx2$>的。Multiplexor 又固定另一个信息，如字符串的长度和校验和，来进一步保证正确的解码。然后 Demultiplexor 校验控制信息并路由字符串到<tx2$>输出。

注意：Multiplexor 的输出字符串的实际形式是：{2 字节的头 (<hdr/list>的 "hdr" 部分)} + {单字节字符串代码(<hdr/list>的 "list" 部分) + { <tx*N*$>的字符串的长度，用字节表示} + {<tx*N*$>} + {信息包中已存在的所有字节的单字节校验和}

参见 Serial Multiplexor (Special)

11、Serial Gather

快键名称：gather

信号/参数：1 个连续输入：<in$>
1 个连续输出：<out$>
2 个参数：<delimiter> 和 <length>

描述：Serial Gather 元素在它找到由<delimiter>参数指定的字符后才求出它的连续输入的值，然后传送字符串已界定的部分（包括<delimiter>）。剩下的字符串片断储存在内部缓冲器中，大小与<length>参数相同。如果输入能够包含多重界定的字符串的话，每一个界定了的部分将按顺序传送直到元素再也找不到有效的<delimiter>为止。

缓冲器<length>至少应该与最长的字符串一样长，但不应超过字符串长度的 254 个字节。如果元素收到超过<length>特征的字符串，而又没有发现<delimiter>的话，就会出现缓冲器溢出的现象。错误信息 "Memory Overflow in Gather" 将被传送到控制台，缓冲器将被清零。

Serial Gather 元素在获取数据碎片方面很有用，例如，接收从 COM 口传送来的数据，再将数据整合后重新发出。传送数据到某些元素时是很危险的，如传送到 Serial Substring 元素上。

参见 Serial Substring

12、Serial I/O

快键名称：stringio, strio, sio

注意：为清楚起见，以下的信号/参数列表以及描述将元素分成输入和输出两个部分。实际上，元素既可以接收数据又可以传送数据。

信号：

连续输入信号：
1 个连续输入：<rx$>
1 个可选数字输入：<**enable**>
任意数量的数字输出：<**o1**> 到 <**o*M***>
每一个输出都有一个对应的参数：<str1> 到 <str*M*>

连续输出信号：
任意数量的数字输入：<**i1**> 到 <**i*N***>
1 个可选数字输入：<**enable**>

每一个输入都有一个对应的参数：<str1> 到 <str*N*>
1 个连续输出：<tx$>

参数：

连续输入/输出参数：1 个可选参数：<delimiter>

描述：

连续输入信号：

在连续输入信号形式中，如果一个给出的输出信号的对应<str>参数与<rx$>输入相匹配，Serial I/O 元素就驱动这个输出信号升高。

在启动时，元素放置一个内部缓冲器，它的大小与最长的<str>参数相同。缓冲器是“后进先出”，因而用新的数据替换出旧的数据。当新的数据进入到 Serial I/O 元素中，它就被加入到缓冲器中并且所有的输出信号被清空（有效地使输出突变（break before make））。然后元素重新核对字符串的所有<str>参数，如果发现一个匹配的，就驱动对应的输出升高。

有时，一行中同样的字符串数据可能进入元素两次，在这种情况下，对应的输出将暂时降低然后再升高。这样就提供了一个可以被逻辑认可的上升沿，此逻辑是由元素驱动的。另外，如果输入包含字符串“HelloWorld”，而两个<str>参数被定义为 "Hello" 和 "World"，对应于 "Hello" 的输出将先被驱动为高电平，然后对应于 "World" 的输出再被驱动为高电平。

可选输入<enable>为高电平时激活元素。当<enable>为低电平时，输入被忽略，输出为低电平。

连续输出信号：

在连续输出信号形式中，元素在信号的上升沿传送那些对应输入信号升高的字符串。

可选输入<enable>为高电平时激活元素。当<enable>为低电平时，所有输入被忽略。如果<enable>变为低电平时有一个输入为高电平，那么这个输入在<enable>重新升高时将被忽略，直到它的下一个上升沿为止。

在输入和输出形式中，可选参数<delimiter>添加指定的特征到所有<str>参数上。例如，如果协议要求：从一个设备处来的和到这个设备的所有字符串要有一个回车/ linefeed，<delimiter>应为 "\n"。

See also Serial Send

13、Serial Memory Dialer

快键名称：srcl, sdial

信号/参数：1 个连续输入：<in$>
1 个数字输出：<dial>
1 个数字输出：<busy>
12 个数字输出：<0> 到 <9>, <*> 和 <#>
1 个双精度参数：<time_betwn_chars> (参见 Numeric Formats)

描述：Serial Memory Dialer 元素和 ASCII Serial Decoder 元素的操作相同，除了 ASCII Serial Decoder 元素的输出可以用任何标准 ASCII 码表示，而 Telephone Dialing Keypad 的输出用数字表示。

Serial Memory Dialer 元素通常与 Telephone Dialing Keypad 元素和 Serial RAM 元素一起工作，如下：Telephone Dialing Keypad 元素产生一个静态字符串。这个输出是联系在 Serial RAM 元素（字符串储存在内存里）的<in$/out$>输入和 Serial Memory Dialer 元素的<in$>输入上的。

Serial RAM 元素的<dial>输出驱动 Serial Memory Dialer 元素的<dial>输入。当<dial>升为高电平时，<in$>被从内存中取出，并且 Serial Memory Dialer 元素的对应数字输出升为高电平，每次一个按顺序进行，直到字符串为 "dialed" 为止。当字符串为 dialed 时<busy>输出升高。

参数<time_betwn_chars>控制拨号操作的速度并指定从一个输出降低到下一个输出升高的时间。同一时间内只有一个输出为高电平。

参见 Serial RAM, Telephone Dialing Keypad, ASCII Keypad, ASCII Serial Decoder

14、Serial Multiplexor (Special)

快键名称：smtx

信号/参数：任意数量的连续输入：<**tx1$**> **到** <**tx*N*$**>

1 **个连续输出：**<**rx$**>

1 **个参数：**<**hdr/list**>

描述：*multiplexor* **使复合信号能够沿着单一的路径被传送，而不丢失任何单独信号的整数部分。原始信号在接收端由** *demultiplexor* **重组。**

在 SIMPL **里，**Serial Multiplexor (Special)**元素通过一个单一的输出路径来传送复合字符串，而** Serial Demultiplexor (Special)**元素重组原始字符串。**

注意：既然这两个元素是相互对称的，那么这两个元素的参数也必然是严密吻合的。

<**hdr/list**>**参数可以用两种方法来区别字符串：一种是为每一个字符串指定一个头；另一种是确定每一个字符串的位置（从** 1 **到** N**）。**Multiplexor **把这两条信息加到每一个字符串上，而** Demultiplexor **利用这个信息将字符串分开并路由它们到相应的输出上。**

"hdr"**部分指定了一个** 2 **字节的头，这个头可以是以下四个字符中的任意一个：**4*, --, AB **或** \x45\x33**。**"list"**部分由每一个字符串中的一个字符组成。也就是说，给出五个字符串，（**<**tx1$**> **到** <**tx5$**>**），**"list"**部分应该是由任意五个字符组成，如：**ABCDE, qwert, **或** \x43\x23\x9A\x21\x33 **等等。因此，拥有五个字符串的**<**hdr/list**>**参数看起来应该如下所示：**

```
4*ABCDE
--qwert
AB\x43\x23\x9A\x21\x33
```

以上面的第一个例子来说，如果字符串 "TEST" **得到值为**<**tx2$**>**，**Multiplexor**将在字符串前加头**"*4" **和字符**"B"**作为前缀，字符是对应于**<**tx2$**>**的。**Multiplexor **又固定另一个信息，如字符串的长度和校验和，来进一步保证正确的解码。然后** Demultiplexor **校验控制信息并路由字符串到**<**tx2$**>**输出。**

注意：Multiplexor **的输出字符串的实际形式是：**{2 **字节的头** (<**hdr/list**>**的** "hdr" **部分**)} + {**单字节字符串代码**(<**hdr/list**>**的** "list" **部分**) + { <**tx*N*$**>**的字符串的长度，用字节表示**} + {<**tx*N*$**>} + {**信息包中已存在的所有字节的单字节校验和**}

参见 Serial Demultiplexor (Special)

15、Serial Substring

快键名称：mid$, sub$

信号/参数：1 **个连续输入：**<**in$**>

1 **个连续输出：**<**out$**>

2 **个参数：**<**pos**> **和** <**length**>

描述：Serial Substring **元素求出它的连续输入信号的值，然后取出并传送字符串中由**<**pos**> **和** <**length**>**参数定义的的部分。这些参数可以用以下两种形式表现：**

第一种形式，<**pos**>**指定第一个被传送字符的位置被传送，而**<**length**>**给出要传送的所有字符的数量。第一个字符的位置由左到右计算，即如果**<**pos**>=1**，被取出的子字符串由左边的第一个字符开始。如果**<**pos**>=2**，被取出的子字符串由左边的第二个字符开始，依此类推。唯一的例外是当**<**pos**>=0

时，子字符串由最右边的字符开始。

第二种形式，<pos> 和 <length>用十六进制格式 *xxyy*h 指定特殊情况来界定字符。这里的 *xx* 表示出现的字符而 yy 表示字符的 ASCII 码。例如，要寻找第三个出现的字母“P”(ASCII 码为十六进制数 50)，参数<pos>应为 0350h。要传送字符从这一点到第二个出现的相连字符（ASCII 码为十六进制数 2D）上，<length>应为 022Dh。

这两种形式不能被混合。也就是说，当<length>指定一个字符的出现时（第二种形式），<pos>不能被指定为一个绝对的字符位置（第一种形式），相反亦是如此。

注意：由第二种形式产生的子字符串不包括界定的字符。同样，<length>参数是由<pos>参数计算得出，而不是输入字符串的开始。因此，就如例子中描述的，子字符串将以第二个由<pos>指定的字符的连字号结束。

应用举例：

形式一：

一个安全的系统以如下格式传送数据：

ZONE：MSTR BED STATUS：FAULT<CR>

要取出并传送区名 "MSTR BED"，<pos> = 6 而 <length> = 12（宽度固定）。要取出并传送区的状态，<pos> = 18 而 <length> = 5。（在这种情况下，区名后将跟有空格。）

形式二：

一个 COM 口以如下格式传送字符串：

TEXT 3-Back in the High Life||Winwood, Steve||Rock\x0D\x0A

应用程序需要把歌曲的名字，艺术家的名字以及流派分成三个独立的字符串进行传送。这就要求用一个 Serial Gather 元素控制三个 Serial Substring 元素。

1、 从 COM 口把字符串和作为<delimiter>参数的 \x0A (line feed) 路由到一个 Serial Gather 元素。

2、 传递字符串 "gathered" 到 Serial Substring symbol 元素，并从第一个出现的连字号字符（ASCII 码为十六进制数 2D）开始至第一个出现的（|）字符（ASCII 码为十六进制数 7C）为止提取数据。因此<pos> =012Dh 而<length> = 017Ch。这样就传送了字符串 "Back in the High Life" 。

3、 传递字符串 "gathered" 到第二个 Serial Substring 元素，并从第二个出现的（|）字符至第一个出现的（|）字符间提取数据。因此<pos> = 027Ch 而 <length> = 017Ch。这样就传送了字符串 "Winwood, Steve"。

4、 传递字符串 "gathered" 到第三个 Serial Substring 元素，并从第四个出现的（|）字符至第一个回车（ASCII 码为十六进制数 0D）间提取数据（ASCII 码为十六进制数 0D）。因此<pos> = 047Ch 而 <length> = 010Dh。这样就传送了字符串 "Rock" 。

比较 Serial Gather

16、Serial Pacer

快键名称：pace$, op117

信号/参数：1 个连续输入：<in$>

1 个连续输出：<out$>

2 个双精度时间参数：<initial_delay> 和 <time_betwn_bytes> (参见 Numeric Formats)

描述：Serial Pacer 元素以由<initial_delay> 和 <time_betwn_bytes>参数指定的速度传播它的连续输入到输出上。输入字符串的第一个特征在由参数<initial_delay>指定的一个延迟后被传送，而参数<time_betwn_bytes>给出了字符传输间的空隙时间。（这个值通常用滴答声表示）

注意：Serial Pacer 元素将中断字符串传送过程中的常规逻辑处理，这可能会引起系统速度的显著减慢。一些连续传送硬件，如 CNXCOM-2 ，有一个固定的步长值，而在这种情况下，步长值将通过硬件来

控制而不是通过软件。

17、Serial Send

快键名称：send

信号/参数：1 个数字输入：<**trig**>

1 个连续输出：<**out$**>

1 个参数：<**string**>

描述：Serial Send 元素在<**trig**>的每一个上升沿传送由<**string**>参数指定的文本。

在应用程序中，当多重字符串被传送时，推荐使用 Serial I/O 元素的输出形式，作为一种有效的可选择项来使用多重 Serial Send 元素。

注意：如果多重 Serial Send 元素的输出积压在一起而它们相应的<**trig**>输入同时升高，元素就维持串联（不需要 Serial Concatenation 元素）。

18、Serial to Analog

快键名称：rxa, s/a, stoa, op103

信号/参数：1 个连续输入：<**rx$**>

任意数量的模拟输出：<**byte1**> 到 <**byte*M***>

任意数量的参数：<**p1**> 到 <**p*N***>

注意：<**p**>参数列表可以通过 Alt+ 命令扩展。

描述：当 Serial to Analog 元素找到一个与由<**p**>参数定义的字符串（或字符串片断）完相匹配者时，它就求出它的连续输入信号的值。然后它从字符串中取出选中的字符并以单独模拟信号的形式传送每一个字符，这样每一个输出信号的值和取出字符的 ASCII 值相同。

等待输入的字符串必须有固定的长度和已知的格式，每一个<**p**>参数必须对应于输入字符串的一个字节。也就是说，如果字符串是由 8 个字节组成，就必须有 8 个<**p**>参数（<**p1**> 到 <**p8**>）。这些参数用以下三种方式进行定义：

1. **<p*N*> = 0000h** 表示字符是不相关的，可以被忽略。
2. **<p*N*> = 01*xx*h** 表示字符必须和 ***xx*** 匹配。例如，如果第三个字符为字母“A”（ASCII 码为十六进制数 41），那么<**p3**>就为 0141h。
3. **<p*N*> = 0200h** 确定一个取出的字符并传送。

应用举例：

假设是一个安全的系统，以如下格式传送数据：

\x2A {单字节区数字} {2 字节区状态} {5 字节区名字} \x35 {单字节区校验和} \x26

在这里，十六进制数为常数而括号内的值为可变值。假设当区数等于 \x01 时，表示区的状态的 2 字节必须被取出并路由到 2 个模拟输出信号上。并假设区的名字和校验和不相关。

既然这里有 12 字节的输入数据，就必须有 12 个<**p**>参数，以如下方式定义：

<**p1**> (必须和 \x2A 匹配): 012Ah

<**p2**> (必须和 \x01 匹配): 0101h

<**p3**> (被取出并路由到<**byte1**>): 0200h

<**p4**> (被取出并路由到 <**byte 2**>): 0200h

<p5> (忽略): 0000h
<p6> (忽略): 0000h
<p7> (忽略): 0000h
<p8> (忽略): 0000h
<p9> (忽略): 0000h
<p10> 必须和 \x35 匹配): 0135h
<p11> (忽略校验和): 0000h
<p12> (必须和 \x26 匹配): 0126h

这里的<p3> 和 <p4>，表示区状态的 2 字节，将被取出并路由到<byte 1>和<byte 2>模拟输出上去。

注意：Serial to Analog 元素将数据放置在输出信号的低字节中。

参见 Serial Gather, Serial I/O, Serial Substring

19、Telephone Dialing Keypad

快键名称：sdac2, telepad

信号/参数：3 个数字输入：<clear>, <enable> 和 <backspace>
12 个数字输入：<0> 到 <9>, <*> 和 <#>
1 个连续输出：<out$>
1 个参数：<#chars>

描述：当<enable>为高电平时，Telephone Dialing Keypad 元素从它的数字输入中产生一个静态的字符串。就这点来说，它和 ASCII Keypad symbol 元素的操作是相同的，除了 ASCII Keypad symbol 元素的输入可以用任何一个标准 ASCII 码来表示，而 Telephone Dialing Keypad 元素的输入用数字表示。

注意：输出信号的静态特性允许它可以通过 Serial RAM 元素被储存在内存中。

在启动时，Telephone Dialing Keypad 元素放置一个内部缓冲器，此缓冲器的大小和<#chars>参数相同。（<#chars>参数等价于 ASCII Keypad 元素的<#chars>参数。）如果进入更多的数字，超过了<#chars>参数指定的范围，额外的数字将被忽略。

因为每一个数字输入升高，对应的特征积聚在缓冲器中，而进入的字符串被重新传送。例如，如果输入字符串 "6532" ，元素将先传送 "6"，然后传送 "65"，接着是 "653"，最后是 "6532"。

<clear>输入移走当前积聚的字符串并传送一个空字符串。<backspace>输入的操作就像一个退格键，使最终用户不需要清除全部序列也可以进行编辑。

当<enable>为低电平时，所有的输入信号被忽略（包括<clear> 和 <backspace>）。如果在<enable>为低电平时仍有积聚的字符串，那么这个字符串将保留在缓冲器中，在<enable>为高电平时被重新传送。

参见 ASCII Keypad, Serial RAM, Serial Memory Search, Telephone Dialing Keypad w/o Backspace

20、Telephone Dialing Keypad w/o Backspace

快键名称：sdac

信号/参数：2 个数字输入：<clear> 和 <enable>
12 个数字输入：<0>到<9>，<*> 和 <#>
1 个连续输出：<out$>
1 个参数：<#chars>

描述：Telephone Dialing Keypad w/o Backspace 元素直接作用于 Telephone Dialing Keypad（电话数字键区），只是没有<backspace>输入。
对大多数应用程序来说，这个元素被认为是无用的，因为如果不用到 Telephone Dialing Keypad 元素的<backspace>输入的话，它可以被简单的赋值为 0 信号。
参见 Telephone Dialing Keypad

七、Sequencing operations

1、Button Presser

快速键名：presses, presser, press
信号：1 个数字输入：<trig>
任意数目的数字输出：<o1>到<o*N*>
描述：Button Presser 信号驱动所有它的输出在<trig>的上升沿升高，在后边缘降低。用这种方法，它可以用来同时触发多个按钮。

2、Stepper

信号/参数：1 个数字输入：<trig>
1 个数字输出：<busy>
任意数目的数字输出
每一个输出都有 2 个对应的单精度参数：<delay> 和 <len> (*参见 Numeric Formats*)
描述：Stepper 信号在其对应的<delay>终止后驱动它的输出信号在<trig>的上升沿升高。每一个输出在由对应参数<len>指定的期间内一直保持为高。任何在<trig>中后来的改变都无效直到所有的输出重新变低。
如果任一个输出升高，输出<busy>也升高；如果所有输出都降低，输出<busy>才降低。

八、Time/Date

1、Clock Driver

快速键名：clock, device system
信号/参数：1 可选连续输出：<tod$>
1 个可选参数：<dst>
描述：Clock Driver 信号使控制系统的内部时钟对其他系统变成可用的。另外，它的单独存在将同步在触摸面板上的内部时钟与控制系统的内部时钟。（输出信号和参数不需要被定义）
参数<dst>指定了可能格式的三分之一：0 = none；1=标准的白天存储时间；2=南半球白天存储时间。
参见 Serialize Date，Past，When

2、Serialize Date

快速键名：date$

信号/参数：1 **个连续输入：**<**tod$**>

1 **个可选数字输入：**<**init**>

1 **个连续输出：**<**date$**>

1 **个参数：**<**format**>

注意：输入<**tod$**>**通常是由** Clock Driver **信号驱动的。**

描述：Serialize Date **信号通过它的**<**tod$**>**输入收到当前的日期，并产生此年中的天日期，用由**<**format**>**参数指定的串行数据格式来表示。如下：**

<format>	<date$>
1d	December 1, 2000
2d	Dec 1, 00
3d	01-Dec-00
4d	12/1/00
5d	1.12.00
6d	01 Dec 00
7d	01/Dec/00

输出串在 1）**启动时更新，**2）**在午夜更新，**3）**在**<**init**>**的上升沿更新，**4）**或当时间和日期由** Set System Clock **信号、**ViewPort**、**1 **个在** CNMSX-Pro **前面板上的手动调节或** Crestron Module **的**"Set Time/Date"**设定时更新。**

参见 Clock Driver**，**Ser System Clock**。**

3、Past

信号/参数：1 **个连续输入：**<**tod$**>

任意数目的数字输出：<**o1**> **到** <**o*N***>

对每一个输出，都有一个对应的参数：<**time1**> **到** <**time*N***>

注意：输入<**tod$**>**通常由** Clock Driver **信号驱动，或（在以前生产的** Cresnet **系统中）由** CNSMPTE **卡驱动。**

描述：Past **信号通过它的连续输入**<**tod$**>**收到当前时间，并驱动它的每一个输出信号在由相应的**<**time**>**参数指定的一天中的某个时间升高。**

参数< **time**> **的格式为** xx.xx.xx.xx**，这** 4 **段把参数**< **time**>**分为小时，分钟，秒和毫秒。例如，要表示** 3:25 PM **又** 26 seconds **（下午** 3 **点** 25 **分** 26 **秒），**<**time**>**应该为** 15.25.26.00s**。一旦有** 1 **个输出升高，只要**<**time**>**表示的时间比**<**tod$**>**早,它将一直保持为高。因为**<**tod$**>**通常由** Clock Driver **信号驱动，这就表明当此天中的时间变成** 00.00.00.00 **时，这个输出将在午夜之前一直保持为高。**

参见 Clock Driver**。**

4、Set System Clock

快速键名：set_clock

信号：4 个数字输入：<**hour+**>, <**hour-**>, <**min+**> 和 <**min-**>
描述：Set System Clock 信号在相应的输入信号的每一个上升沿通过增加或减少小时和分钟的设置来调节控制系统的内部时钟。
注意：内部时钟也可以通过以下方式来调节：1）用 ViewPort；2）手动调节 CNMSX-Pro 或 CNRACKX 前面板上的设置；3）用 Crestron 模块上的"Set Time/Date"。

5、When

信号：1 个连续输入：<**tod$**>
任意数目的数字输出：<**o1**> 到 <**o*N***>
对每一个输出，都有一个对应的参数：<**time1**> 到 <**time*N***>
1 个双精度参数：<**pulse width**>
注意：输入<**tod$**>通常由 Clock Driver 信号驱动，或（在以前生产的 Cresnet 系统中）由 CNSMPTE 卡驱动。
描述：When 信号通过它的<**tod$**>连续输入收到当前时间,并在由参数<**time**>指定的某一时刻驱动每一个输出信号输出高电平。
参数< **time**> 的格式为 xx.xx.xx.xx，这 4 段把参数< **time**>分为小时，分钟，秒和毫秒。例如，要表示 3:25 PM 又 26 seconds（下午 3 点 25 分 26 秒），<**time**>应该为 15.25.26.00s。
在由<**pulse width**>指定的期间内输出将保持为高，当<**pulse width**>失效或下降时输出为低。
参见 Clodk Driver 。

九、Timers

1、Delay

信号/参数：1 个数字输入：<**trig**>
1 个可选数字输入：<**reset**>
任意数目的数字输出：<**o1**> 到 <**o*N***>
每一个输出都有一个对应的单精度参数：<**delay1**> 到 <**delay*N***>（参见 Numeric Formats）
描述：在对应的<**delay**>停止后，Delay 信号驱动每一个输出信号达到<**trig**>输入的水平。注意到所有指定的延迟都是相对独立的，也就是说，没有累积的延迟效果。
一旦<**reset**>为高，可选输入<**reset**>就立即驱动所有的输出达到<**trig**>的水平（无延迟）
参见 Serialize Date，Past，When

2、Debounce

信号/参数：任意数目的数字输入：<**i1**> 到 <**i*N***>
每一个输入都有一个对应的数字输出：<**o1**> 到 <**o*N***>
1 个双精度参数：<**time**>（参见 Numeric Formats）
描述：如果<**time**>保持输入状态值不变至失效时，Debounce 信号就驱动 1 个给出的输出信号达到

它对应的输入的水平。

例如，如果<time>为 5 秒而且给出的输出为低，它对应的输入必将升高并在输出变高之前保持 5 秒。然后这个输出保持为高直到它的输入变低并保持 5 秒为低。

每一个输入有一个对应的输出，每一个输入输出对之间是相对独立的。在输出对应的输入状态呈现出来之前，启动时，在由<time>指定的期间内，所有的输出都为低。

示例：当 1 个按钮开关对应一个数字输入时，Debounce 信号可以用在软件中。通常，当“按”按钮被按下或放开时，在高电平与低电平的过渡的极短时间内，有一个快速的机械反弹。带有一个短<time>参数的 Debounce 信号，比如 0.5 秒，可以用于确保信号在消失于申请之前的稳定。

3、Multiple One Shots

快速键名：mmv, m1shot, mos

信号/参数：任意数目的数字输入：<i1> 到 <1*N*>

1 个可选数字输入：<reset>

每一个输入，从<i1> 到 <i*N*>，都有一个对应的数字输出：<o1> through <o*N*>

1 个双精度参数：<pulse_time>（参见 Numeric Formats）

描述：Multiple One Shots 信号，正如它的名字一样，将几个独立的 One Shots 信号进行群组为一个单一的信号，所有信号共享同样的<pulse_time>。每一组输入/输出表现为 One Shot 符号的<trig> 和 <out>信号（在 Multiple One Shots 符号中没有<trig*>或 <out*>信号）。

Multiple One Shots 符号在输出信号对应的输入的上升沿驱动指定的输出信号在由<pulse_time>指定的期间内达到高，当<pulse_time>停止时达到低。在<pulse_time>期间内输入的任何改变都不会复位<pulse_time>，也不会影响输出。当<pulse_time>停止而输出变低时，它可以在对应输入的另一个上升沿被重新触发。

当<reset>保持为高时，可选输入<reset>立即使所有的输出为低。

注意：如上所述，这个符号在需要多中断信号的软件中非常有用，它们共享同样的中断时间。然而，Multiple One Shots 符号没有<trig*> 和 <out*>接线端，当用来在信号的下降沿产生一个中断或产生补充输出状态时， One Shot 符号是一个更好的选择。

参见 One Shot，Retriggerable One Shot

4、One Shot

快速键名：mv, 1shot, os

信号：1 个数字输入：<trig>

2 个可选数字输入：<trig*> 和 <reset>

1 个数字输出：<out>

1 个可选数字输出：<out*>（<out>的补码）

1 个双精度参数：<pulse_time>（参见 Numeric Formats）

描述：在<trig>的上升沿（或<trig*>的后边沿），One Shot 符号驱动它的输出信号在由<pulse_time>指定的期间内升高，在<pulse_time>停止后降低。在<pulse_time>期间内，<trig> 或 <trig*>中并发的改变不会将<Time>复位，也不会影响输出。当<pulse_time>停止而输出降低后，这个符号就可以被另一个<trig>的上升沿（或的后边沿)触发。

当<reset>保持为高时，不管<trig>或<trig*>的状态如何，可选输入<reset>立即强迫输出变低。

如果用到了输入<trig*>，输出将在<reset>的后边沿升高，并一直保持直到<pulse_time>停止。如果<reset>已被定义，那么<trig*>就必须被定义。如果<trig*>对于应用程序来说不是必需的，它可以被赋值为 0。

注意：<out> 和可选信号<out*>不能突变。这意味着当<out>的状态改变时，<out>和<out*>都将在瞬时拥有同一个值。

参见 Multiple One Shots，Retriggerable One Shot

5、Oscillator

快速键名：osc

信号：1 个数字输入：<gate>

1 个数字输出：<out>

2 个双精度参数：<hi_time> 和<lo_time>（参见 Numeric Formats）

描述：Oscillator 符号驱动它的输出在由<hi_time>指定的期间内升高，在由<lo_time>指定的期间内降低，在<gate>为高时，持续在这两个状态之间振荡。这种振荡开始于<gate>的上升沿。当<gate>变低时，输出立即变低。

6、Pulse Stretcher

快速键名：hmv, pstretch, ps

信号/参数：1 个数字输入：<trig>

1 个数字输出：<out>

1 个可选数字输出：<out*>（<out>的补码）

1 个双精度参数：<delay_time>（参见 Numeric Formats）

描述：一旦<trig>输入为高，Pulse Stretcher 符号驱动它的输出信号达到高。在<trig>降低后，输出在由<Time>指定的额外期间内仍然保持为高（除非是<trig>升高，否则<Time>参数不会影响输出）。

7、Retriggerable One Shot

快速键名：rmv, r1shot, ros

信号/参数：1 个数字输入：<trig>

2 个可选数字输入：<trig*> 和 <reset>

1 个数字输出：<out>

1 个可选数字输出：<out*>（<out>的补码）

1 个双精度参数：<pulse_time>（参见 Numeric Formats）

描述：在<trig>的上升沿（或<trig*>的后边沿），Retriggerable One Shot 符号驱动它的输出信号在由<pulse_time>指定的期间内达到高。

与 One Shot 符号不同 Retriggerable One Shot 符号认可任何并发的<trig>的上升沿(或<trig*>的后边沿)，甚至当<out>为高而引起它重新计算时也是如此。输出将不会变低直到全部的<pulse_time>持续时间无打断的全部过去为止。

一旦<reset>保持为高，不论<trig> 或 <trig*>的状态如何，可选输入<reset>都立即使输出

变低。

如果 <trig*>输入被用到，输出在<reset>的后边沿将升高并保持为高直到<pulse_time>停止。如果<reset>已被定义，那么<trig*>就必须被定义。如果应用软件中不需要<trig*>，它可以被赋值为 0。

注意：<out>和可选<out*>信号不会突变。这表示当<out>的状态改变时，<out>和<out*>都将在一瞬时拥有同样的值。

参见 One Shot，Multiple One Shot

十、System Control

1、Intersystem Communications

快速键名：xsig, iscomm, isc

注意：为了清晰起见，以下的信号/参数列表将符号分成了输入和输出两部分。实际上，符号可以同时接收和发送数据。

信号：连续输入形式：

1 个连续输入：<rx$>

任意数目的模拟或连续输出：<aout1> 到 <aout*N*>

任意数目的数字输出：<dig_out1> 到 <dig_out*N*>

连续输出形式：

1 个连续输出：<tx$>

任意数目的模拟或连续输入：<ain1> 到 <ain*N*>

任意数目的数字输入：<dig_in1> 到 <dig_in*N*>

参数： 2 个参数：<Offset> and <Option>

描述： Intersystem Communications (ISC) 符号使数据能通过合适的连接口，如 Modem,Ethernet,RS-422,在两个或多个控制系统间来回传输。

ISC 符号可拥有任意数量的数字、模拟和连续输入和输出的组合。无论何时，任一个它的输入值发生了改变，符号都会将这一信息编码为一个字符串，并通过<tx$>输出传送出去。当然，在 1 个逻辑波内发生的信号转换越多，传送的字符串越长。

典型的<tx$>与连续驱动上的输入有关，例如 CNXCOM 卡，它将数据传送到 COM 口。在接收端，编码的连续数据通过 1 个二级 ISC 符号（一般经由 COM 口和连续驱动器）的输入传进来；这个二级符号将数据进行解码并驱动它的输出达到相应的值。

（对于一个 X-generation 控制系统，并没有对在 1 个逻辑波内传唤信号（可被传送/接收）的数量做实际的限制。而在 ST-CP 和 CN-系列的控制系统内，只能进行 120 个数字转换或 60 个模拟转换。）

一个 ISC 符号的输入与另一个 ISC 符号的输出成映射，如下：所有的信号被位置限制在内，由 0 开始。第一个定义的信号的指数是 0，第 2 个的指数是 1，第 3 个是 2，依此类推。既然信号和指数相对应，输入/输出对就不一定要取相同的名字。也就是说，1 个指数为 4 的输入信号总是对应于 1 个有相应指数的输出信号，不论信号的名字是什么。

参数<Option>决定了符号将怎样处理同一信号在 1 个逻辑波内的多重转换。对此参数来说，唯一有效的值是 0、1 和 2，在大多数情况下，我们推荐用 1 作为设定值。

当<Option>=0，只有当转换改变了信号的值的时候，符号才传播信号的此次转换。例如，如果 ISC 符号收到如下数据：

<digital 1 high>, <analog 2 = 1234>, <digital 1 low>, <analog 2 = 2345>

此符号不会理会<digital 1>的升高或<analog 2>的值 1234，它只会处理<digital 1 low> 和 <analog 2 = 2345>的逻辑。

值为 1 的<**Option**>激活了符号去处理同一信号的多重转换。就如例子中描述的，符号将首先处理<digital 1 high>和 <analog 2 = 1234>的逻辑，然后才处理<digital 1 low>和 <analog 2 = 2345>。这种格式将用于 Ethernet 应用，在那里，潜在的问题可以导致在越过 1 个网络传送数据时的不规律现象。

最终，当<**Option**>=2 时，元素传播出所有的数据，即使同一信号收到多次相同的值也是如此。（这和其他形式不同，它们只是传播改变的值。）如果元素收到如下数据：

<digital 1 high>, <analog 2 = 1234>, <digital 1 low>, <analog 2 = 1234>

元素将传播<analog 2>的值两次。模拟信号驱动 1 个“可重复触发”(“retriggerable”)元素比如 Serial/Analog One-Shot 的情况下，可应用这种形式。当 2 个 ISC 元素被定义为同等的情况下，并有拥有信号名相同的模拟输入和输出时，此形式不适合使用。

限制： 在 Ramping 操作中，模拟信号的值也许不断的变化，Ramping 操作可导致缓冲器过度使用，当通过 ISC 元素时还会引起数据丢失。基于这种情况，可用 Analog Value Sample 和 Oscillator 元素来控制数据更新。

ISC 元素不能识别 Serial Buffer 元素的输出。在这种情况下，当 1 个 Serial Buffer 必须驱动一个 ISC 时，Serial Send 元素的输出将和 Serial Buffer 的输出联系起来，而 ISC 将识别出数据是真正连续的。设置 Serial Send 的<**string**>参数为空（“”），设置它的<**trig**>输入为 0。

有时，1 个 ISC 元素必须驱动多个 ISC 元素。此时，利用<**Offset**>参数来保证数据的正确路由就是很必要的。<**Offset**>包指定的值加入到信号传送队列中；在接收时，这些值被减去。如果减法操作导致了 1 个负值，数据被忽略。例如，想以下 1 个远程 ISC 元素必须和两个本地 ISC 元素互相作用。

远程系统

ISC #1：<**Offset**>=0；250 个已定义的输入和 250 个已定义的输出。

主系统

ISC #2：<**Offset**>=0；200 个已定义的输入和 200 个已定义的输出。

ISC #3：<**Offset**>=200；50 个已定义的输入和 50 个已定义的输出。

假设有 2 个输入信号在 ISC #1 的队列中的 21 和 246 偶然发生转变。数据将传送到 ISC #2 和 ISC #3。

ISC #2，及一个 0 的偏移量，不会从信号的队列中减去任何值。它识别在队列中第 21 信号转换并驱动它的第 21 个输出信号到达相应的值。不过，它忽略了队列中第 246 信号的转换，这是因为它超过了它定义的 200 个输出的范围。

ISC #3 及 1 个 200 的偏移量，从信号队列中减去了这个值。从 21 中减去 200 产生了 1 个负数，数据被忽略。从 246 中减去 200 后，元素识别队列中第 246 信号的转变并驱动自己的第 46 个输出信号达到相应的值。

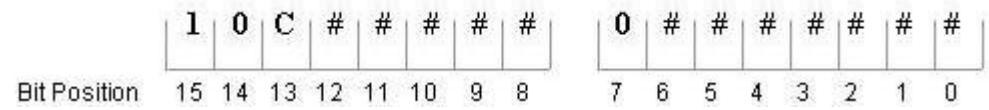
为了能够更深入的在多个 ISC 元素间进行通讯，SIMPL Windows 提供了 2 个能够同步传送/接收数据元素的初始化命令：**Clear Outputs** (\xFC) and **Send Status** (\xFD)。这两个命令通常由 1 个 Serial Send 元素直接发到 1 个本地 ISC 的<**rx$**>输入或 1 个远程 ISC 的<**tx$**>输出上。

当这个 ISC 元素收到 Clear Outputs 命令时，它强制所有的输出为 0；当它收到 Send Status 命令时，它求出它的输入的值并传送所有没有被设置为 0 的数字或模拟信号。注意，连续数据不能通过 Send Status 命令更新。

通讯形式

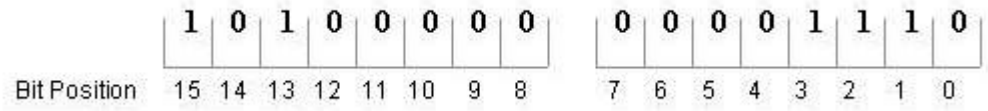
（以下的描述完全用以做对比；不是一定要这么做才能够理解编码方式以便在程序中使用元素。）

数字数据用 2 **字节（**16 **位）来编码，如下：**

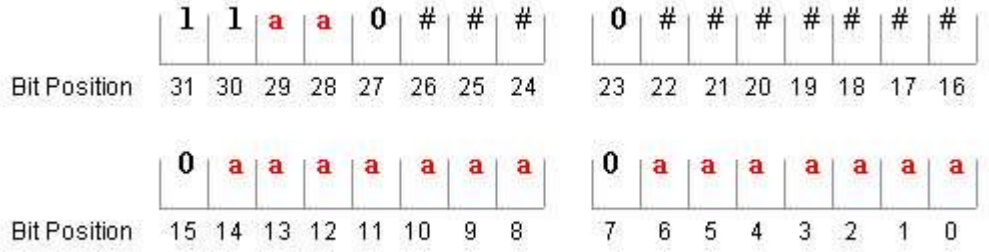


当第 15 **位（最重要的一位）被设为** 1 **而第** 14 **位被设为** 0**，表示** 1 **个新数字数据的开始。第** 13 **位设为信号状态的补数。也就是说，如果信号为高电平，**C=0**；如果信号为低电平，**C=1**。第** 7 **位（**2 **字节数的高位）设为** 0**。第** 12 **位，信号的指针（用** #**号标识），越过第** 1 **个和第** 2 **个字节被展开，还带着用于填充的第** 1 **个** 0**。**

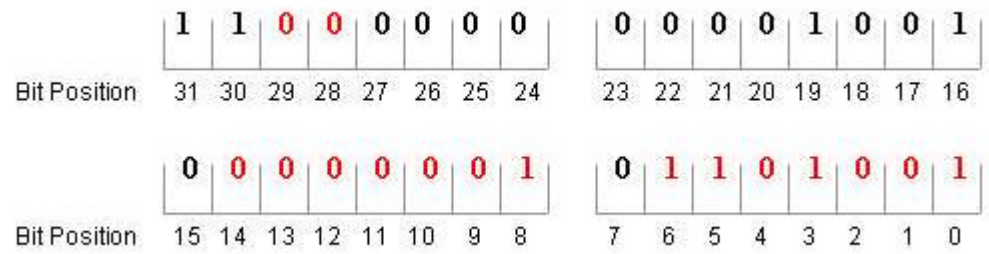
例如，认为 1 **个在指针** 14 **处（二进制数** 0000 1110**）的数字信号变成低电平。就会出现以下的形式，数据将这样被发送：**



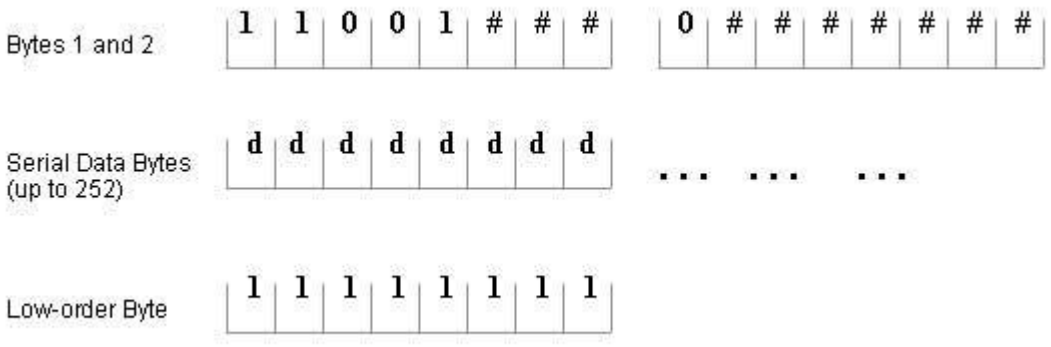
模拟数据用 4 **字节（**32 **位）形式进行编码，如下：**



当第 31 **为（最重要的位）和第** 30 **位都被设为** 1 **时，表示一个新模拟数据的开始。第** 27 **位设为** 0**，就像第** 23 **位、第** 15 **位和第** 7 **位（每个字节的高位）一样。**16 **位模拟信号的值（由** 1 **个红色的字母“**a**”标识）越过** 1**、**3**、**4 **字节被展开，还带着用于填充的第一个** 0**。信号的第** 10 **位指针（用** # **号标识）越过** 1**、**2 **字节展开，带着用于填充的第一个** 0**。例如，认为** 1 **个在指针** 9 **处（二进制数** 0000 1001**）模拟信号的值为** 233**（二进制数** 1110 1001**）。就出现以下形式，数据将这样被发送：**



数字信号和模拟信号用一种固定长度的格式来编码，与它们不同，连续数据可以用不同长度的格式来编码，如下：



当 5 **个高顺序位被设为** 1-1-0-0-1，**表示新连续数据的开始。字节** 2 **的高顺序位设为** 0。**信号的第** 10 **位指针（用** # **号标识）越过字节** 1 **和字节** 2 **展开带着用于填充的首位的** 0。**连续数据（用字母** d **标识）的** 252 **个字节都如此。最终，低顺序位字节被设为** FFh（**二进制数** 1111 1111），**表示连续数据的结束。**

参见 Serialize Date，Past，When

Communications Format

2、Intersystem Communications w/Status Req

快速键名：xsig3, iscommsr

注意：为了清楚起见，以下的信号/参数列表将符号分成了输入和输出两部分。实际上，符号可以同时接收和发送数据。

信号：连续输入形式：

1 **个连续输入：**<**rx$**>

任意数目的模拟或连续输出：<**aout1**> **到** <**aout*N***>

任意数目的数字输出：<**dig_out1**> **到** <**dig_out*N***>

连续输出形式：

1 **个连续输出：**<**tx$**>

任意数目的模拟或连续输入：<**ain1**> **到** <**ain*N***>

任意数目的数字输入：<**dig_in1**> **到** <**dig_in*N***>

参数： 2 **个参数：**<**Offset**> and <**Option**>

描述：除了 Intersystem Communications **符号把瞬时发生的变化立即传送给输入外，** Intersystem Communications with Status Request **符号与** Intersystem Communications **符号的操作相同。与之相反，** ISC w/Status Request **符号只有在已收到“发送状况”（**Send Status**）指令时才传送数据。**

“发送状况”指令(\xFD**)通常由** Serial Send **符号发给**<**rx$**>**输入——**1 **个本地的** ISC **或发给**<**tx$**>**输出——**1 **个远程** ISC**。当** ISC w/Status Req **符号收到这个指令，它求出它的输入值并传送所有不为** 0 **的数字和模拟信号。注意，连续数据不能被处理或通过** ISC w/Status Req **符号传送。**

参见 Intersystem Communications

3、Hard Reset

快速键名：reset2, hreset
信号：1 个数字输入：<**hard reset**>
描述：Hard Reset 符号在<**hard reset**>的上升沿预初始化上载到控制系统程序。这等同于在视窗（Viewport）中按下<F10>键，或在视窗的“功能”菜单（Viewport Function menu）中选择 Hard Reset 指令。任何没有保存在永久内存中的程序将被删除。

4、Soft Reset

快速键名：reset1, sreset
信号：1 个数字输入：<**soft reset**>
描述：Soft Reset 符号在的上升沿预初始化当前程序。这等同于在视窗中按下<F9>键，或在视窗的“功能”菜单（Viewport **Function** menu）中选择 **Soft Reset** 指令。这个符号通常与包含很多时间元素的程序一起使用，这是为了将程序放置在一个可知的状态中。

Numeric Formats

Numeric values can be expressed in a number of formats, where the character in parentheses represents the format identifier:

- *(d)ecimal*
- *(h)exadecimal*
- *(%) percentage*
- *(s)econds*
- *(t)icks (1 tick = 1/112.5 seconds)*
- *(')character(') (single byte)*

The allowable range of analog values expressed in each format is as follows:

Format	*Minimum*	*Maximum*
Decimal	*0d*	*65535d*
Hexadecimal	*0h*	*FFFFh*
*Percentage**	*0%*	*100%*
*Seconds***	*0s*	*582.53s*
Ticks	*0t*	*65535t*
Byte	*' ' (space, ASCII 20h)*	*'~' (tilde, ASCII 7Eh)*

**Percentage and seconds formats can be expressed with precision of .01% or .01s.*
***Double precision time values range from 0.0 seconds to 19,088,743 seconds.*

Every parameter has a default format if none is specified when the symbol is defined.

Break before Make

The property of some symbols that forces one output low before any other output can go high. This behavior is useful when the outputs of a symbol are used to enable a number of other symbols. Typically, it is important for one symbol to be disabled before another becomes enabled.

Static/Transient Data

Static data maintains its value in memory until something in the application drives it to a new value (or until power is removed from the system). In SIMPL, all digital and analog signals are static, as are the serial outputs of the ASCII and Telephone Dialing Keypad symbols.

Transient data is temporary and decays over a period of time, which can range from milliseconds to seconds, depending on the application. In SIMPL, all serial signals are transient (except for the Keypad symbols just described).

NVRAM

Non-volatile random-access memory: a region of memory in the control system that retains information written to it when power is switched off.

High/Low Bytes

High byte: In a 2-byte grouping representing a 16-bit value (bits 0 through 15), the byte containing the most significant bits (bits 8 through 15).

Low byte: The byte containing the least significant bits (bits 0 through 7).