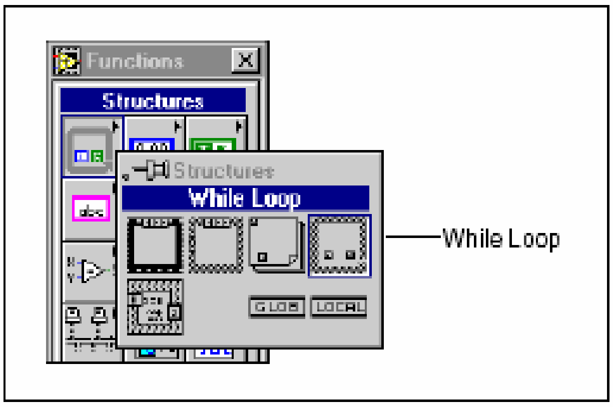
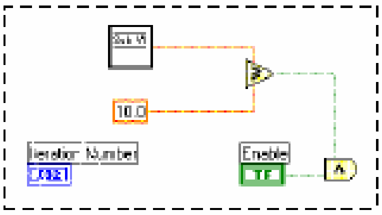
A. While Loop

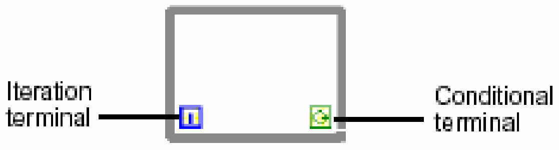
A *While Loop* repeats part of your block diagram code multiple times. You place a While Loop in the block diagram by first selecting it from the **Structures** subpalette of the **Functions** palette.



Then use the mouse cursor to click-and-drag a selection area around the code you want to repeat. When you release the mouse button, a While Loop boundary *encloses* the code you have selected as shown below.



The completed While Loop is a resizable box. You can add additional block diagram elements to the While Loop by dragging and dropping them inside the boundary with the mouse.



Conditional terminal

Iteration terminal

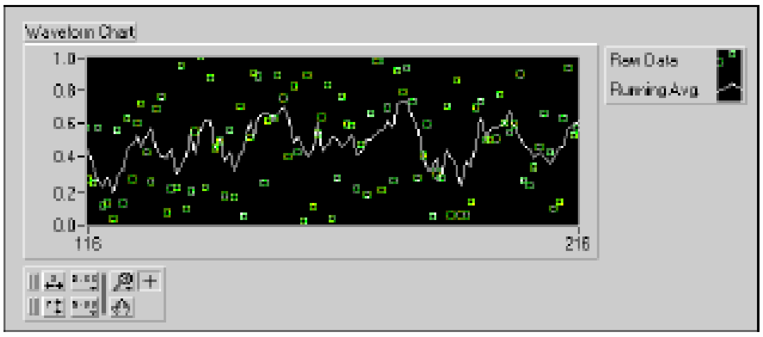
The VI repeats the code inside the While Loop until the Boolean value passed to the *conditional terminal* (an input terminal) is FALSE. The VI checks the conditional terminal at the *end* of each iteration; therefore, **the While Loop always executes once**. The *iteration terminal* is a numeric output terminal that contains the number of times the loop has executed, starting at zero. (That is, during the first execution of the loop, the iteration terminal contains the number zero.)

A While Loop is equivalent to the following pseudo-code:

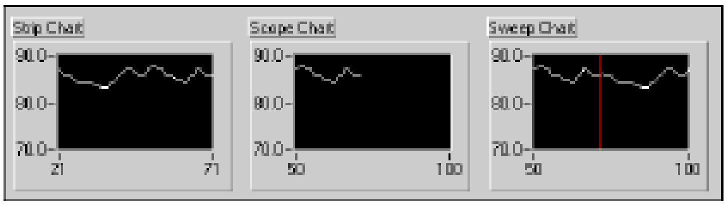
```
Do
Execute Diagram Inside the Loop (which sets the condition)
While the condition is TRUE
```

B. Waveform Charts

The waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is in the **Graph** subpalette of the **Controls** palette. Waveform charts may display single or multiple traces. An example of a multiple-plot waveform chart is shown below.



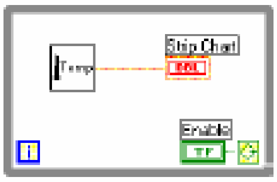
The waveform chart has three update modes—*strip chart*, *scope chart*, and *sweep chart*. You can select the update mode by popping up on the waveform chart and choosing one of the options from the **Data Operations » Update Mode** menu. (In run mode, select **Update Mode** from the chart's pop-up menu.)



The *strip chart* has a scrolling display similar to a paper strip chart. The *scope chart* and *sweep chart* have retracing displays similar to an oscilloscope. Because there is less overhead in retracing a plot, the scope chart and the sweep chart are significantly faster than the strip chart in displaying plots. On the scope chart, when the plot reaches the right border of the plotting area, the plot is erased, and plotting begins again from the left border. The sweep chart acts much like the scope chart, but the display does not go blank when the data reaches the right border. Instead, a moving vertical line marks the beginning of new data and moves across the display as new data is added.

Wiring a Single-Plot Chart

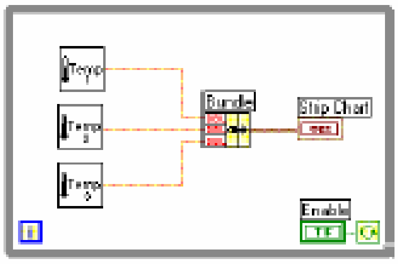
You can directly wire a scalar output to a waveform chart. The data type displayed in the waveform chart's terminal icon will match the input type, as shown in the example below.



Wiring a Multiple-Plot Chart

Bundle function

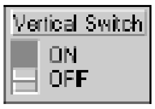
Waveform charts can accommodate more than one plot. You must bundle the data together using the **Bundle** function (**Cluster** subpalette). In the example below, the **Bundle** function "bundles" or groups the output of the three different VIs that acquire temperature for plotting on the waveform chart. Notice the change in the waveform chart terminal icon. To add more plots, simply increase the number of **Bundle** function input terminals by resizing the **Bundle** function using the Positioning tool.



Mechanical Action of Boolean Switches

You may notice that each time you run the VI, you first must turn on the vertical switch and then click on the Run button. With LabVIEW, you can modify the mechanical action of Boolean controls. The choices for the mechanical action include: Switch When Pressed, Switch When Released, Switch Until Released, Latch When Pressed, Latch When Released, and Latch Until Released.

For example, consider a vertical switch shown below. The default value of the switch is off (FALSE).



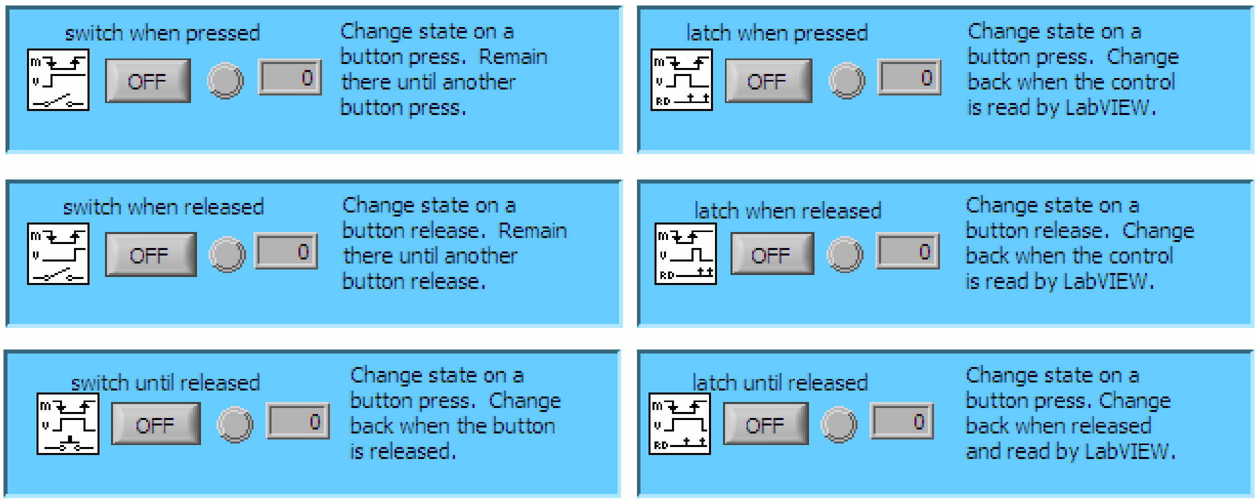
Switch When Pressed action changes the control value each time you click on the control with the Operating tool. The action is similar to that of a ceiling light switch, and is not affected by how often the VI reads the control.

Switch When Released action changes the control value only after you release the mouse button during a mouse click within the control's graphical boundary. The action is not affected by how often the VI reads the control.

Switch Until Released action changes the control value when you click on the control and retains the new value until you release the mouse button, at which time the control reverts to its original value. The action is similar to that of a door buzzer, and is not affected by how often the VI reads the control.

Latch When Pressed action changes the control value when you click on the control and retains the new value until the VI reads it once, at which point the control reverts to its default value. (This action happens whether or not you continue to press the mouse button.) This action is similar to that of a circuit breaker and is useful for stopping While Loops or having the VI do something only once each time you set the control.

Latch Until Released changes the control value when you click on the control and retains the value until your VI reads the value once or until you release the mouse button, whichever occurs last.



switch when pressed — OFF — 0 — Change state on a button press. Remain there until another button press.

latch when pressed — OFF — 0 — Change state on a button press. Change back when the control is read by LabVIEW.

switch when released — OFF — 0 — Change state on a button release. Remain there until another button release.

latch when released — OFF — 0 — Change state on a button release. Change back when the control is read by LabVIEW.

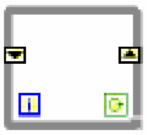
switch until released — OFF — 0 — Change state on a button press. Change back when the button is released.

latch until released — OFF — 0 — Change state on a button press. Change back when released and read by LabVIEW.

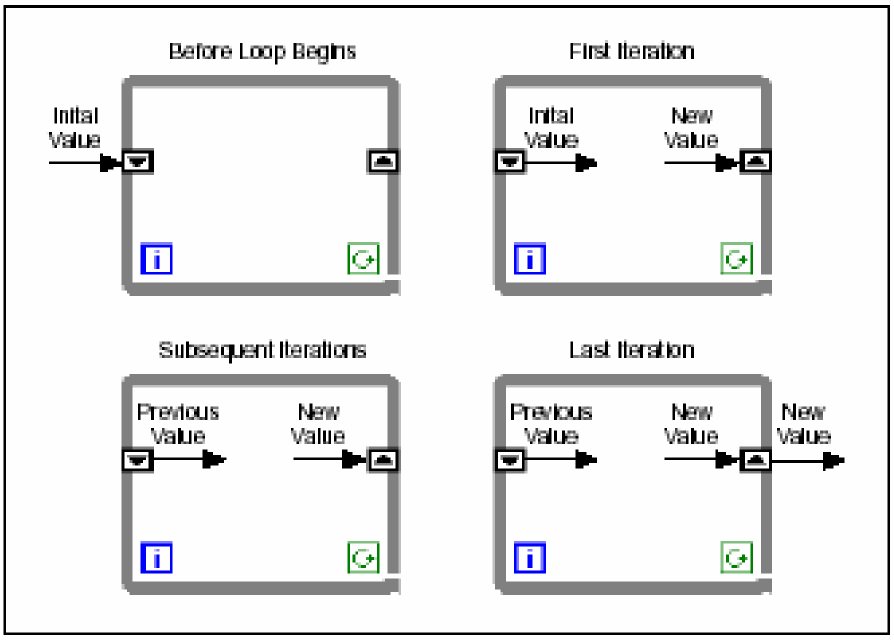
C. Shift Registers

You use shift registers (available for While Loops and For Loops) to transfer values from one iteration to the next. You create a shift register by popping up on the left or right loop border and selecting **Add Shift Register** from the pop-up menu.

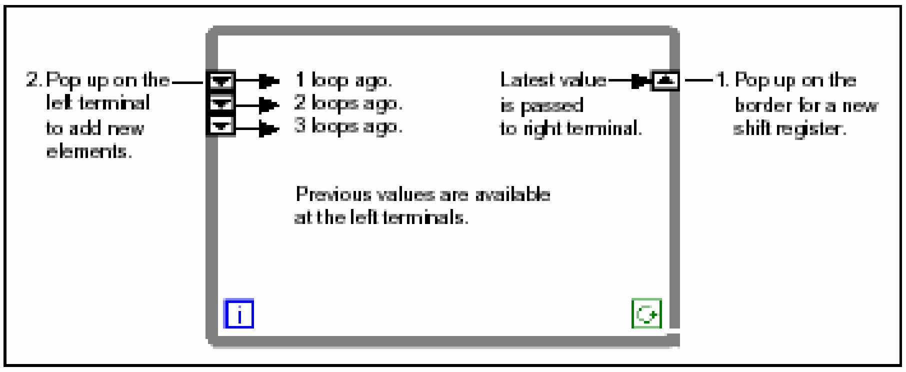
The shift register contains a pair of terminals directly opposite each other on the vertical sides of the loop border.



The *right* terminal stores the data on the completion of an iteration. That data is shifted at the end of the iteration and it appears in the *left* terminal at the beginning of the next iteration (see the figure below). A shift register can hold any data type—numeric, Boolean, string, array, and so on. The shift register automatically adapts to the data type of the first object wired to the shift register.



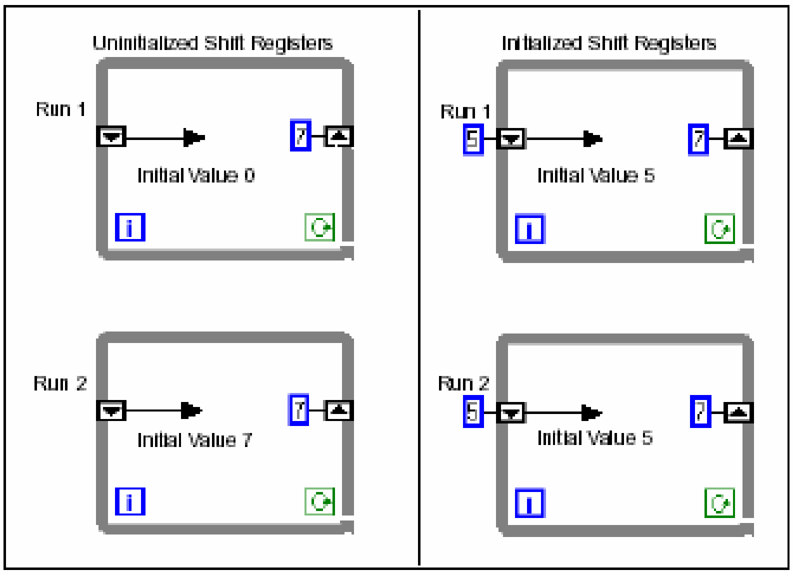
You can configure the shift register to remember values from several previous iterations. This feature is very useful when you are averaging data points. You create additional terminals to access values from previous iterations by popping up on the *left* terminal and choosing **Add Element** from the pop-up menu. For example, if you add two more elements to the left terminal, you can access values from the last three iterations.


2. Pop up on the left terminal to add new elements.
1 loop ago.
2 loops ago.
3 loops ago.
Latest value is passed to right terminal.
1. Pop up on the border for a new shift register.
Previous values are available at the left terminals.

Initializing Shift Registers

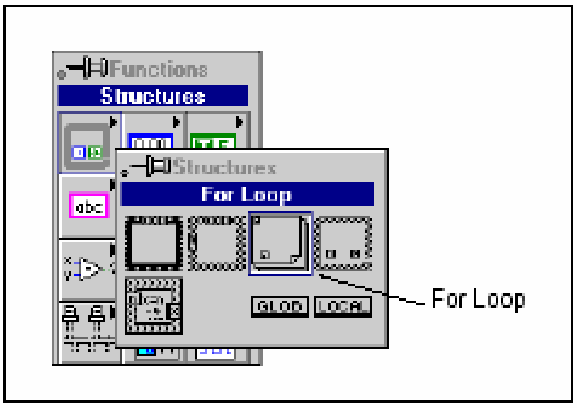
To initialize the shift register with a specific value, wire the initial value to the left terminal of the shift register (outside the loop). If you leave the initial value unwired, the initial value will be the default value for the shift register data type. For example, if the shift register data type is Boolean, the initial value will be FALSE. Similarly, if the shift register data type is numeric, the initial value will be zero.

Note ***LabVIEW does not discard values stored in the shift register until you close the VI and remove it from memory. In other words, if you run a VI containing uninitialized shift registers, the initial values for the subsequent run will be the ones left from the previous run.***



D. For Loop

A For Loop repeats part of your block diagram code a predetermined number of times. You select a For Loop from the **Structures** subpalette of the **Functions** palette, and then enclose the code you want to repeat in the For Loop boundary. A For Loop (shown below) is a resizable box. The For Loop has two terminals: the count terminal (an input terminal) and the iteration terminal (an output terminal). The count terminal specifies the number of times to execute the loop. The iteration terminal contains the number of times the loop has executed.

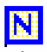


The difference between the For Loop and the While Loop is that the For Loop executes a *predetermined* number of times. A While Loop stops repeating the code it encloses only if the value at the conditional terminal becomes FALSE. The For Loop is equivalent to the following pseudo-code:

```
For i = 0 to N-1
Execute Diagram Inside The Loop
```

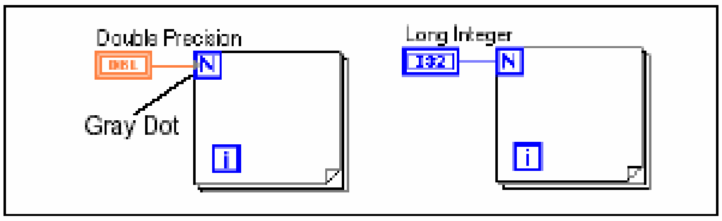
Numeric Conversion

Until now, all the numeric controls and indicators you have used are double-precision floating-point numbers. LabVIEW, however, can represent numerics as integers (byte, word, or long) or floating-point numbers (single, double, or extended precision). If you wire together two terminals that are of different data types, LabVIEW will convert one of the terminals to the same representation as the other terminal. As a reminder, LabVIEW places a dot, called a coercion dot, on the terminal where the conversion takes place.



For Loop count terminal

For example, consider the For Loop count terminal. The terminal representation is long integer. If you wire a double-precision floating-point number to the count terminal, LabVIEW converts the number to a long integer. Notice the gray dot in the count terminal of the first For Loop.


Double Precision
DBL
N
Gray Dot
i
Long Integer
I32
N
i