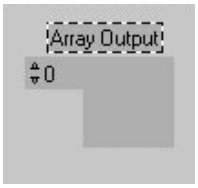
PHY 406 - Microprocessor Interfacing Techniques
LabVIEW Tutorial - Part VIII
Arrays and Clusters

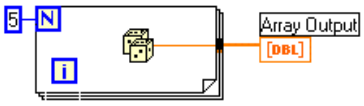
Arrays, Clusters and Conglomerates

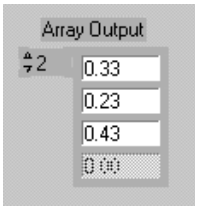
LabVIEW has two group constructs: arrays which are N-dimensional arrays of entities and clusters which in any other language would be called "structures". Both of these are quite useful and are also essential for accessing some of the more powerful LabVIEW functions.



We have already met arrays briefly in an example above. This VI makes up a 1-D array of random numbers 5 long. We examine this array by using the array shell from the **array** controls. Notice that at the moment this has no reality to it - it is just a shell because we don't know what sort of thing to display.

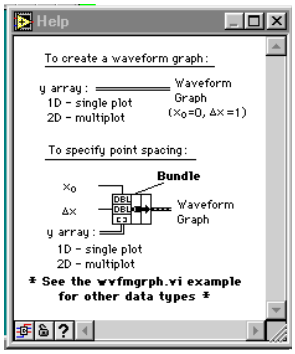
To get further we drop a numeric indicator into the shell - now we know it will be a numeric array (as opposed to an boolean array for example. The shell now looks like this. The left-hand control is for the index - the right-hand value is the value of that index. Since we haven't actually wired the indicator up yet, we don't know whether the index "0" is valid or not (although it should be) so the value is "greyed out" even though I changed the background colour to white.





You can use the select tool to resize the indicator box to display more elements and when you have wired up the indicator to a real array and run the program, the indicator comes to life. Notice that since the array size is 5 and array indices start from 0, the last element on the list doesn't exist and is therefore still greyed out.

Now let's feed the array to a graph (remember charts are for values which come one at a time, graphs are for arrays). The result should be a simple graph of the value of the random number generator for five values each time you run the VI.

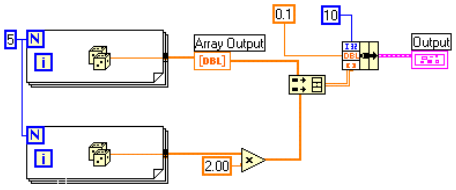


A useful technique to learn at this stage is to "highlight execution" which allows you to run the VI in slow motion and watch the values being produced. This is controlled by the light-bulb icon. Press the light-bulb and run the VI - you can now see each value being produced and how the loop runs. You may need to try it a couple of times to see the full effect. Press the lightbulb again to turn this off.

The x-axis of the graph is the array index. It is possible to change that by adding information to the input and for this purpose we need a

cluster. Since we don't have one we are going to have to make one by "bundling" a number of elements together. Look very carefully at the help information for the graph. It shows that we can have up to three elements in the cluster: the initial x, the x increment and the array. In order to use these we must bundle up the array and two other scalars into a cluster.

Choose the **Cluster>>Bundle** function. It appears on the block diagram as a two-element bundler. Use the select tool to resize it to three elements. Wire the output to the graph and the third input to the array. Generate two constants (using the wiring tool pop-up on the terminal and select **Create Constant**). Use 10 for the initial x and 0.1 for the increment. Now when you run the VI you should be able to see that the x-axis has been rescaled appropriately.



More study of the help for the graph indicates that a 2-D array will yield two plot lines. So now we can duplicate the array generator (select the For-loop and do a **Ctrl-drag**). Just to distinguish between the outputs - put a multiplier on one of them to double the array value. The multiplier copes with multiplying an array by a scalar without further intervention. Since we have two 1-D arrays and we need a single 2-D array we need the **Array>>Build Array** function resized to two inputs. We feed a 1-D in each input and a 2-D becomes the output. We can then feed this to the bundle function and then on to the graph. If this all works then the graph should now have two lines on it and an x-scale from 10.0 to 10.4. Here is my block diagram for this.

You might be interested to run this VI under the "execution highlight". Notice that both the for loops execute together. This is a consequence of dataflow programming. Since neither of the loops is waiting for an input, both are permitted to run within the limits of available CPU time. LabVIEW has built-in parallelism.

Summary

- LabVIEW has both arrays (single and multi-dimensional) and clusters
- Clusters are simialr to structures in other computer languages
- Arrays can be built automatically or explicitly using array builders. The array builders will combine arrays as well as scalars
- Clusters are built using the **Bundle** directives (they can also be **Unbundled)**

Exercise

Write a VI to generate a graph of a user-supplied quadratic over a user-supplied range of values.