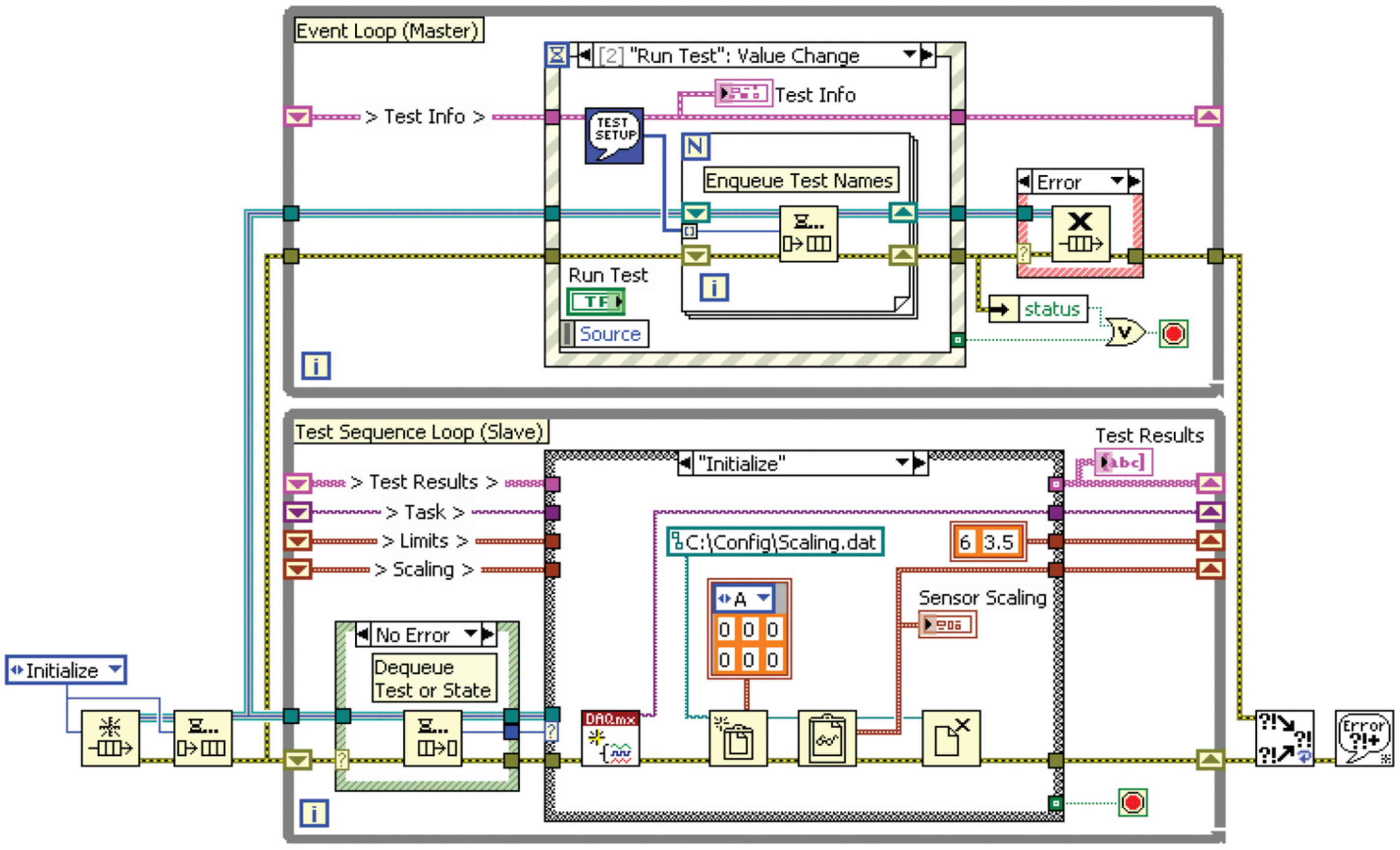
NATIONAL INSTRUMENTS LabVIEW



The LabVIEW Style Book

Peter A. Blume

EASE OF USE • EFFICIENCY • READABILITY • SIMPLICITY

PERFORMANCE • MAINTAINABILITY • RELIABILITY

The LabVIEW Style Book

Rule 4.13 Tunnel wires through left and right borders of structures

Rule 4.14 Do not wire through structures unnecessarily

Tunnel wires into structures through their left border, and out of structures through their right border. Avoid tunneling wires through the top and bottom borders. Also avoid passing wires through structures if they are not utilized within the structure, unless their purpose is clearly labeled. It is particularly annoying to flip through many frames of a multiframe structure, such as a Case or an Event structure, searching for places where the data in the wire is modified. However, sometimes it is useful to pass a few spare wires through a Case structure, such as the state machine design patterns that are discussed in Chapter 8. This practice reduces maintenance when additional wires are needed. Be sure to clearly label unused wires as `Unused`, or `Spare`.

Rule 4.15 Never obstruct the view of wires and nodes

- Avoid overlapping diagram objects

Avoid obstructing the view of wires and nodes by overlapping them on the diagram. An occasional crossover of a wire routed horizontally with a wire routed vertically is unavoidable. For example, sometimes it is necessary to wire the iteration terminal of a looping structure, normally located at the bottom left of the structure, to a location above and to the right. Many of these same loops have wires routed horizontally across the entire structure, via either tunnels or shift registers. The error cluster is a prime example. If the iteration terminal is to remain on the bottom left, there may be no choice but to cross over the horizontal wires. The obstruction is minimized if the vertical wire is routed through a location of minimum wire density, overlapping as few wires as possible. Additionally, never overlap a wire with an object or a node. A wire running underneath a function or subVI resembles an input and output to the node.

Rule 4.16 Limit wire lengths such that source and destination are visible on one screen

Rule 4.17 Never use local and global variables for wiring convenience

Rule 4.18 Label long wires and wires from hidden source terminals

Ideally, the source and destination of every wire should be readily visible without scrolling the diagram window. However, this is not always possible, even if the diagram is limited to one screen. For example, the terminals may be hidden within the frames of a multiframe structure. In this situation, be sure to label the wire in the frames or areas where the source terminal is not visible. While limiting wire lengths is desirable, long wires are preferred over no wires. Never use local or global variables as a method of reducing wire clutter. Variables increase processing overhead, memory use, and complexity. Moreover, variables undermine LabVIEW's dataflow principles by obscuring the actual source of the data. When variables are written and read from more than one location on the diagram, it becomes difficult to determine what is actually affecting the data. When wires are used, it is easy to trace the data to its unique source terminal. If the wires are long or the data source terminal is hidden, label them. Indicate the name of the wire's data source so that it is readily apparent when you view the diagram. Use the greater than sign (>) to reinforce the direction of data flow. Considerations with respect to local and global variables are discussed in Section 4.3, "Data Flow."

Rule 4.19 Place unwired front panel terminals in a consistent location

Place any unwired terminals of front panel objects in a consistent location on the diagram so that developers can find them easily. Note that any terminals not contained by a repeating structure are read only once. This is a problem for Boolean controls configured with latching action. In this case, the control is not able to reset itself. If the terminal's control is associated with an event that is registered by an Event structure, place the terminal within the corresponding event case. This ensures that the terminal's value will be read each time the event fires. For unwired terminals not associated with any events, simply place them to the left of the diagram's primary structure.

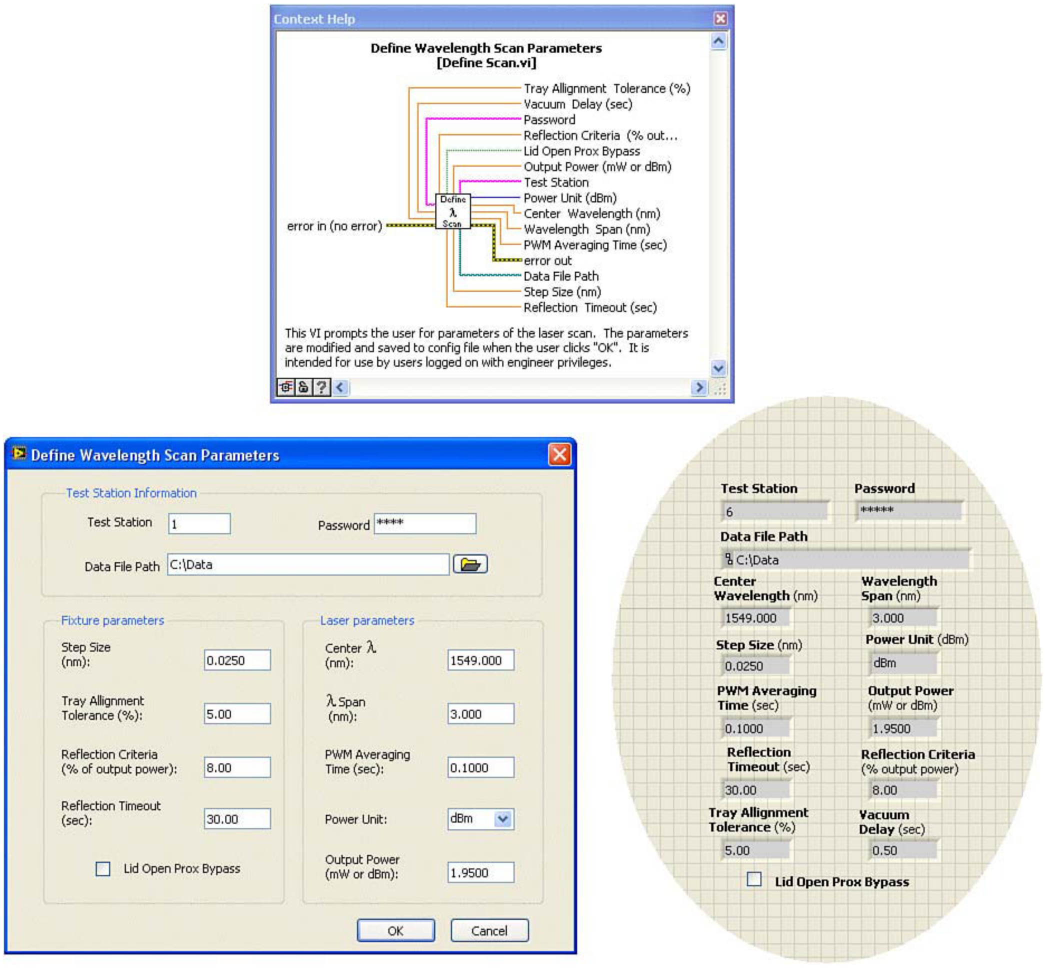
4.2.2 Cluster Modularization

Rule 4.20 Modularize wires of related data into clusters

It is much easier to implement clear wiring techniques and maintain organized diagrams if you reduce the overall number of wires you have to work with. Use clusters to group related data and reduce the quantity of wires. Wherever you have several wires of related data that are needed in the same areas of your diagrams, replace the individual wires with a cluster. This is analogous to modularizing low-level routines into cohesive subVIs. The data elements in the cluster should be related and serve a common purpose.

For example, consider the measurement routine from an optical filter test application, shown in Figure 4-5. The application prompts the user to define the laser scan parameters, configures a wavelength tunable laser source, measures the filter's transmission characteristics, graphs the data, and saves the data to file. Figure 4-5A contains the panel, nonvisible indicators, and Context Help window for Define Scan VI, the dialog used for selecting the laser scan parameters. This is the first subVI called in the measurement sequence shown in Figure 4-5B. Define Scan VI provides multiple parameters used by the subsequent VIs. As shown in the Context Help window, the connector pane is densely populated with individual terminal assignments for each parameter. In Figure 4-5B, there are many individual wires flowing through a relatively simple routine. The subVI connector terminal for Save Scan VI, the last subVI in the dataflow sequence, is also very densely populated. The wiring is kept reasonably neat because of even spacing and judicious terminal assignments. However, much toil is required to achieve this result, and even more toil is required to modify the VI. Specifically, any change to the required measurement parameters entails changes to the wires, subVIs, and connectors throughout the diagram.

Figure 4-5C contains an alternative implementation of Define Scan VI, with the laser scan parameters returned as a cluster. In Figure 4-5D, the measurement routine is revised using the cluster instead of individual wires. Wire clutter and development toil are substantially reduced. Additionally, parameters can be added and removed from the cluster without requiring any changes to the wires and subVI connector terminal assignments. Hence, clusters improve the diagram's appearance while reducing overall development and maintenance effort.



Context Help

Define Wavelength Scan Parameters
[Define Scan.vi]

Tray Allignment Tolerance (%)
Vacuum Delay (sec)
Password
Reflection Criteria (% out...
Lid Open Prox Bypass
Output Power (mW or dBm)
Test Station
Power Unit (dBm)
Center Wavelength (nm)
Wavelength Span (nm)
PWM Averaging Time (sec)
error out
Data File Path
Step Size (nm)
Reflection Timeout (sec)
error in (no error)

This VI prompts the user for parameters of the laser scan. The parameters are modified and saved to config file when the user clicks "OK". It is intended for use by users logged on with engineer privileges.

Define Wavelength Scan Parameters

Test Station Information

Test Station: 1
Password: ****
Data File Path: C:\Data

Fixture parameters

Step Size (nm):	0.0250
Tray Allignment Tolerance (%):	5.00
Reflection Criteria (% of output power):	8.00
Reflection Timeout (sec):	30.00
☐ Lid Open Prox Bypass	

Laser parameters

Center λ (nm):	1549.000
λ Span (nm):	3.000
PWM Averaging Time (sec):	0.1000
Power Unit:	dBm
Output Power (mW or dBm):	1.9500

OK Cancel

Test Station: 6	Password: *****
Data File Path: C:\Data	
Center Wavelength (nm): 1549.000	Wavelength Span (nm): 3.000
Step Size (nm): 0.0250	Power Unit (dBm): dBm
PWM Averaging Time (sec): 0.1000	Output Power (mW or dBm): 1.9500
Reflection Timeout (sec): 30.00	Reflection Criteria (% output power): 8.00
Tray Allignment Tolerance (%): 5.00	Vacuum Delay (sec): 0.50
☐ Lid Open Prox Bypass	

Figure 4-5A
Define Scan VI is a dialog that prompts the user to specify laser scan parameters. It returns 15 parameters through separate connector terminals for each.

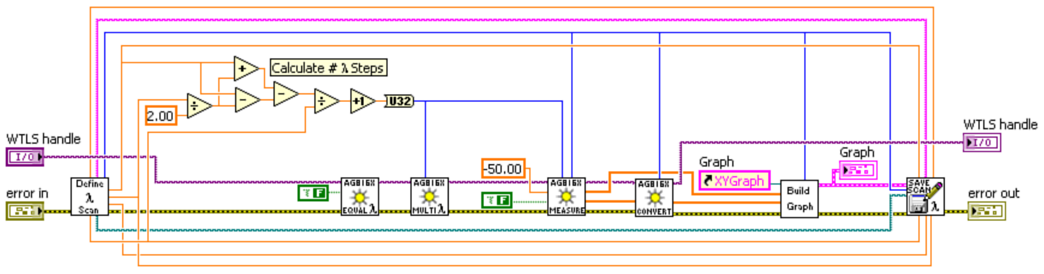
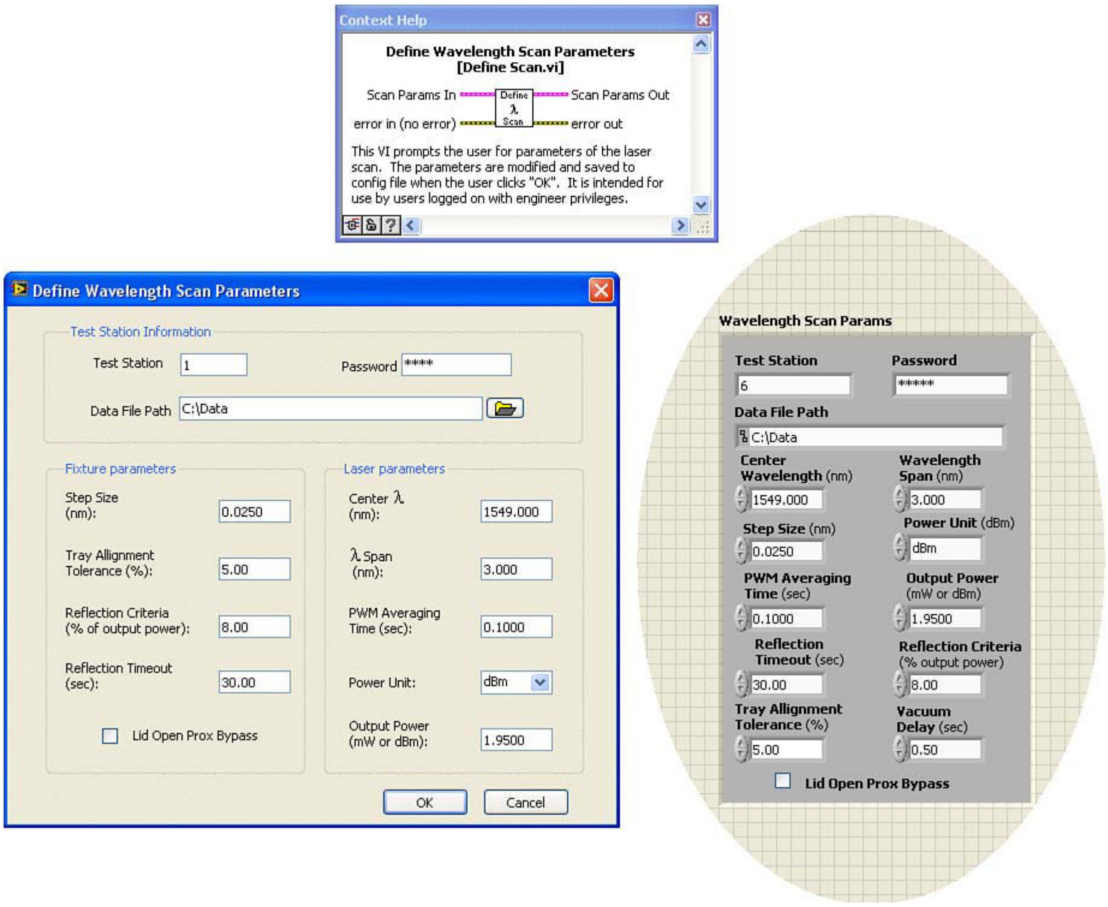


Figure 4-5B
An optical filter measurement routine calls Define Scan VI and propagates the laser scan parameters using individual wires. The subVI connector panes are densely populated, wiring is cluttered, and maintenance is tedious.



Context Help

Define Wavelength Scan Parameters
[Define Scan.vi]

This VI prompts the user for parameters of the laser scan. The parameters are modified and saved to config file when the user clicks "OK". It is intended for use by users logged on with engineer privileges.

Define Wavelength Scan Parameters

Test Station Information

Test Station	1	Password	****
Data File Path	C:\Data		

Fixture parameters

Step Size (nm):	0.0250
Tray Allignment Tolerance (%):	5.00
Reflection Criteria (% of output power):	8.00
Reflection Timeout (sec):	30.00

☐ Lid Open Prox Bypass

Laser parameters

Center λ (nm):	1549.000
λ Span (nm):	3.000
PWM Averaging Time (sec):	0.1000
Power Unit:	dBm
Output Power (mW or dBm):	1.9500

Wavelength Scan Params

Test Station	Password
6	*****
Data File Path	
C:\Data	
Center Wavelength (nm)	Wavelength Span (nm)
1549.000	3.000
Step Size (nm)	Power Unit (dBm)
0.0250	dBm
PWM Averaging Time (sec)	Output Power (mW or dBm)
0.1000	1.9500
Reflection Timeout (sec)	Reflection Criteria (% output power)
30.00	8.00
Tray Allignment Tolerance (%)	Vacuum Delay (sec)
5.00	0.50

☐ Lid Open Prox Bypass

Figure 4-5C
Define Scan VI modularizes the laser scan parameters into a cluster. Terminal assignments are simplified.

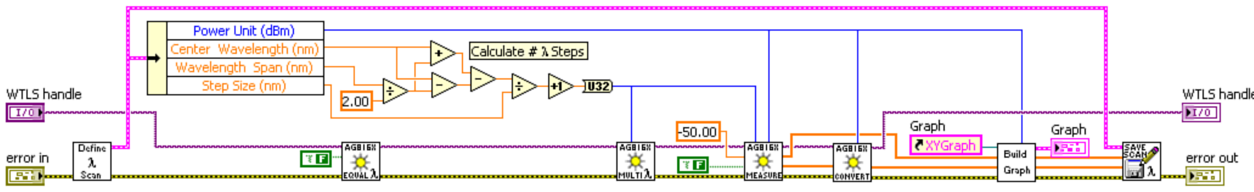


Figure 4-5D
The cluster of laser scan parameters propagates throughout the diagram. Wire clutter is reduced. Parameters can be added and removed from the cluster without affecting wiring and subVI terminals assignments.

In some situations, it makes sense to define a cluster for just two data elements. A small cluster is useful to associate data that is very closely related and is frequently used together. As an example, the high and low limits of a measured parameter may be read from a database, edited by the user in a dialog VI, passed to another routine that compares the measured data to the limits, and passed to additional routines where a report is generated and the data and limits are logged to file. A two-element cluster containing the high and low limits eliminates one wire and logically binds the data together. In this case, the cluster is more beneficial for associating related data than for eliminating wire clutter.

The optical filter measurement routine propagates the wavelength and power data arrays using two separate wires. As shown on the right half of Figure 4-6A, the two wires are used together in most places, with the exception of Convert Power Units VI, which requires only the power data. In Figure 4-6B, the Convert Power Units VI is incorporated as a subVI within Measure VI. Hence, the power data is converted to appropriate units before it is returned. Also, the wavelength and power arrays are modularized into a cluster. Finally, the routine that unbundles several scan parameters and calculates the number of wavelength steps has been modularized into a cohesive subVI.

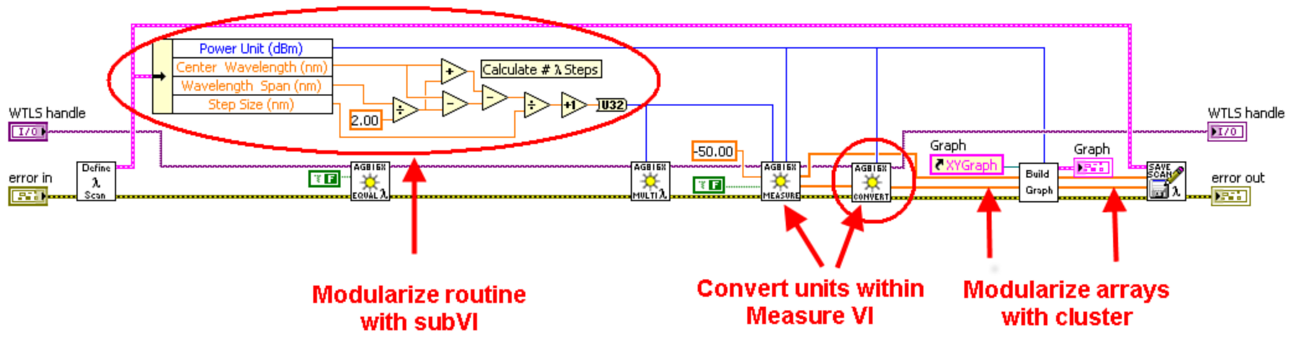
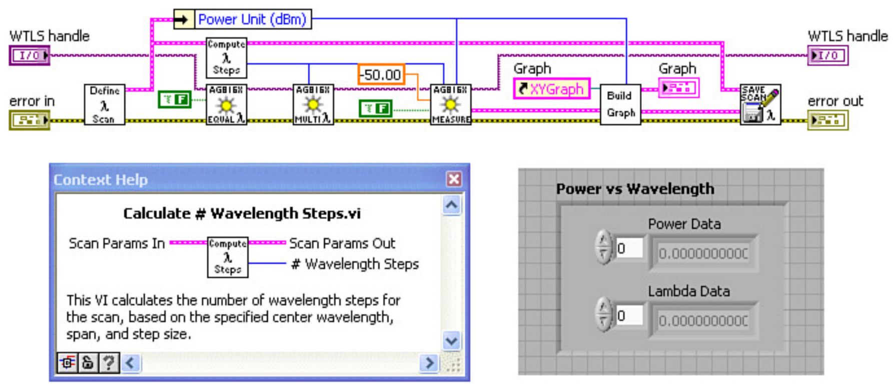


Figure 4-6A
Two separate wires for the wavelength and power arrays are used together in most places and may be modularized into a cluster. The routine for calculating the number of wavelength steps can be modularized into a subVI. The routine for converting power units can be performed within Measure VI.



Calculate # Wavelength Steps.vi

Scan Params In — Scan Params Out — # Wavelength Steps

This VI calculates the number of wavelength steps for the scan, based on the specified center wavelength, span, and step size.

Figure 4-6B
The routine is further modularized with Calculate # Wavelength Steps VI and Power vs. Wavelength cluster.

Rule 4.21 Save clusters as type definitions

I cannot overemphasize this rule. (Consequently, it is further discussed in Chapter 6, "Data Structures.") Save every cluster as a type definition or strict type definition. A **type definition** (or **typedef**, for short) is a control that maintains its data type information in a CTL file. The typedef is copied onto any number of VI panels as a control or indicator, or diagrams as a constant, by either dragging and dropping from the project tree, or choosing either **Select a Control** from the **Controls** palette or **Select a VI** from the **Functions** palette. All instances of the typedef maintain the data type specified in the typedef's CTL file. Therefore, multiple instances of the control are maintained from one location via the Control Editor window. This is extremely beneficial for clusters because most clusters are used in multiple places, and the contents are subject to change. For example, in Figure 4-5C, new parameters are added to or removed from the **Wavelength Scan Params** cluster by simply adding or removing the corresponding controls to the typedef. In Figure 4-5D, all of the subVIs containing the typedef will automatically update to match the revised cluster.

A **strict type definition** (also known as **strict typedef**) maintains the control's properties, in addition to the data type, in the CTL file, so that all instances of the strict typedef maintain an identical appearance and behavior. I prefer the strict typedef because I often find that I need to maintain a common range, default value, and appearance among instances. To specify the type definition status, choose the corresponding item from the Typedef Status ring control on the Control Editor window's toolbar. Clusters and type definitions are discussed in greater detail in Chapter 6.

4.3 Data Flow

This section covers data flow, the fundamental principle of LabVIEW. It contains a brief review of basic principles and style rules, considerations regarding variables and Sequence structures, and techniques for optimizing data flow. The rules are illustrated with a combination of simple diagram snippets and a working application example.

4.3.1 Data Flow Basics

In LabVIEW, **data flows** along wires from source terminals to destination terminals. A block diagram node executes when data is received at all wired input terminals. Upon completion, data is supplied to its output terminals and propagates to the next node in the dataflow path. The dataflow principle distinguishes LabVIEW from traditional text-based software-development environments. The following are some basic rules regarding data flow.

Rule 4.22 Always flow data from left to right

Rule 4.23 Propagate the error cluster