For: dwight
Printed on: Fri, Jul 16, 1993 09:52:26
From book: manual
Document: appc
Last saved on: Thu, Sep 3, 1992 08:36:58

Appendix C: XAssign

XAssign Overview

The conventional way to generate an Assign flowgraph is to write a C program that calls interface routines to generate the flowgraph. Although this allows for maximum flexibility in generating a flowgraph, it requires you to write and compile a new program to generate each new flowgraph.

XAssign, the Graphical User Interface for the Assign signal processing compiler, provides an alternative to Assign's C programming interface. It is designed to provide a more intuitive interface that frees you from the details of programming and command line options and lets you focus on the flowgraph programming model itself.

XAssign's key features are the following:

- intuitive visual interface
- creation and modification of flowgraphs
- support for compilation and execution of flowgraphs
- extensive help facility
- reusable library of standard node definitions

NOTE

XAssign has been designed for full OSF/Motif-style compliance.

Creating an XAssign application

The are four steps in creating a XAssign application:

1. **Write Assign node functions and XAssign node definitions. (optional)**
   The node functions are sequential C functions, instances of which correspond to nodes in the flowgraph. For each node function, you must create a node definition so the new node can be used with the XAssign user interface. If the standard libraries included are adequate for your application, this step is not necessary.
2. **Generate and test subgraphs.**
   Create several XAssign graphs using the XAssign interface and test each one separately. You can compile and test these smaller subgraphs on the host.
3. **Combine and test subgraphs.**
   Once each subgraph has been tested and is working properly, you can combine the subgraphs into one large graph using the copy and paste editing functions. You can now compile and test the complete graph.
4. **Compile and run the final graph.**
   Once you have tested the final graph, you can compile and run the graph on the iWarp system.

The XAssign Window

Figure C-1 shows an XAssign window.

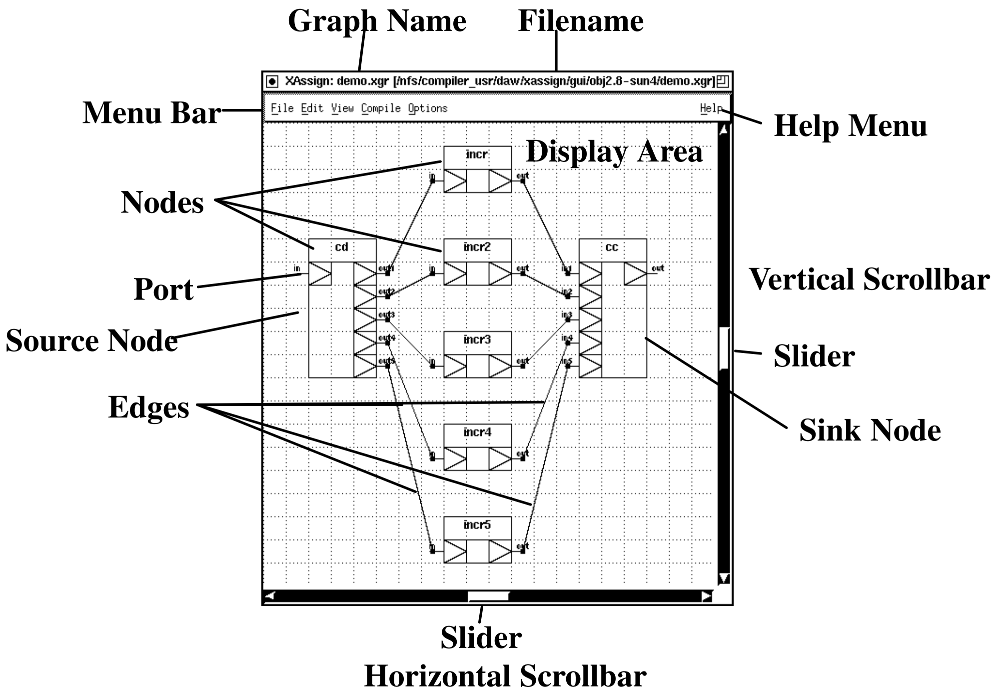

Figure C-1: Example XAssign Window

The following describes the contents of the XAssign window.

- **Menu Bar**
  Allows selection of all XAssign functions.
- **Flowgraph Display Area**
  Shows all or part of the complete flowgraph. The *nodes* and *edges* in this part of the flowgraph are displayed. Each square on the grid is called a *cell*. The coordinates of all XAssign objects are in terms of cell coordinates. Nodes are represented as labeled rectangles, with cells containing arrows representing *ports*. Ports on the left side of the node are always input ports, and ports on the right are always output ports. Edges are represented as lines connecting coordinates, and their purpose is to connect node ports on the graph.
- **Horizontal and Vertical Scrollbars**
  The slider in each scrollbar shows the location the the display area in the complete flowgraph. The location can be changed by dragging the slider, or with mouse or keyboard actions.

In the example graph shown in Figure C-1, the ‘‘cd’’ node is the *source node* because it has no input port connections. The input going into this node is the input to the flowgraph. The ‘‘cc’’ node is the *sink node* because that node has no output port connections. The output going out of this node is the flowgraph output.

The cd node is a column distribute node that divides its input matrix to output one column of the original matrix to each output port. Each column is fed into an ‘‘incr’’ node. Each incr node increments each element in its input vector and outputs the incremented vector as its output vector. The cc node is a column collect node that collects the input vectors into a matrix with each input vector being one column in the output matrix. So, in this example, the result is simply that each element in the input matrix is incremented. This same operation could have been done with a single increment node, but the above graph would be faster for large data sets because the work can be divided between processors.

For more information on the definition of a flowgraph, a node, a port, and an edge, refer to Chapter 2, *The Assign Flowgraph*.

Starting XAssign

To start XAssign, enter the following on the command line:

```
xassign [option ...] [filename ...]
```

option can be any option described in the following section.

filename can be any xassign graph file, and is usually suffixed by the .xgr extension.

XAssign accepts any number of options or filenames. A window is created to display each graph file specified on the command line, or a single empty window appears if no filename is specified.

Command Line Options

XAssign supports all of the standard X Toolkit options plus the following additional options:

-fn	Sets the *fontlist resource instead of the *font resource. To display a list of possible fonts, use the `xlsfonts` command. Note: this is not the standard Xt -fn, but will work similarly.
-font	Same as -fn
-h path	Set the path to the Help Text file, if not the default.

Managing Flowgraphs

XAssign allows editing of several flowgraphs at once and sharing of data between flowgraphs. Flowgraphs are managed using the file menu. The following options are available on the file menu

New	Creates a new XAssign window with an empty graph
Open	Loads a previously saved XAssign graph file.
Save	Saves an XAssign graph file with the current name. If there is no name, this is equivalent to Save As.
Save As	Saves an XAssign graph file with a specified name.
Close	Closes a graph.
Exit	Closes all windows and ends the XAssign session. If any graphs have been modified but not saved, you must confirm closing the window.

Viewing a Flowgraph

The following sections outline the features available to help you view your flowgraphs.

Scrollbars

The scrollbars are located on the bottom and right of the XAssign window. Scrollbars show the area of the flowgraph displayed, and allow you to manipulate the display area with the mouse. The rectangle inside the scrollbar is called the *slider*. The size of the slider in relation to the length of the complete scrollbar indicates the size of the screen displayed in that dimension relative to the graph's entire area. The position of the slider on the scrollbar indicates the position of the display relative to the graph's entire area. You can change the current location displayed with the scrollbar in the following ways.

- **Dragging the slider**
  Hold the left mouse button on the slider, drag the slider to the correct location, then release the mouse button. As you drag the slider, the screen is continually updated to reflect the current location.
- **Clicking or pressing the area above or below the slider on the scrollbar**
  Click the left mouse button above or below the slider to move the display (and slider) one display length (or one slider length) toward the mouse pointer. Press the left mouse button below the slider and hold it down, and the display will repeatedly scroll one slider length toward the mouse pointer.
- **Clicking or pressing the scrollbar arrow**
  Click the left mouse button on an arrow at the end of a scrollbar to move the display one cell in the direction the arrow points. Press the left mouse button on an arrow and hold it down, and the display will scroll one cell at a time in the arrow direction until you release the button.

Keyboard

You can also use the keyboard to move the graph display. The following key sequences allow you to change the display area.

- **Cursor Key**
  Moves the display area one cell in the cursor key direction. This is useful for small changes in location.
- **Shift + Cursor Key**
  Moves the display area one quarter of the length of the display in the cursor key direction. This is useful for browsing the graph for the correct location.
- **Control + Cursor Key**
  Moves the display area to the end of the graph in that direction. This is useful for quickly moving to another part of the graph.
- **Keypad 5 key (between the cursor keys)**
  Centers the display on the center of the graph.
- **Page Up, Page Down, Home, & End**
  The Page Up, Page Down, Home, and End key moves the display area up, down, left, and right, respectively, one display length. This is primarily for quickly moving around in the graph.

Zooming In and Out

Not only can you change the position of the display, but you can also change the amount of detail displayed. This is called "zooming in" and "zooming out." Zooming in displays more detail, but less of the graph is displayed. Zooming out displays less detail, but more of the graph is displayed. When possible, the display retains the same center position.

You can select zooming under the View menu on the pulldown or popup menu as *Zoom In* or *Zoom Out*. The following keyboard shortcuts also perform the zoom operations:

- Keypad + Zoom In
- Keypad - Zoom Out

Toggling the Grid

You can optionally display a grid in the background to help position nodes and edges with the mouse. You can toggle the grid from the View menu on the pulldown or popup menu as **Grid Toggle**. The keystroke for toggling the grid is the keypad Enter key.

Creating and Editing a Flowgraph

Nodes and edges are the basic building blocks of a flowgraph. Creating a node and edge are the first basic steps in flowgraph creation. Other editing functions help in modifying a current graph and also can be used to speed graph creation.

The following terminology for mouse buttons is used throughout explanations that follow:

select	The left mouse button. This button is used to define a selection or the area selected in the display area.
edge-toggle	The middle mouse button. This button is used to toggle edge adding mode.
extend	Clicked by holding down shift and clicking the select button. It is used to extend the current selection region.

The following terms are also used in the explanations:

Node Definition	A node definition is a file containing information about a certain node type. The information in this file allows XAssign to do the task of the node generation function. For more information about node definition files, refer to the section *Node Definition Files*, later in this chapter.
Node Variables	The node variables of a node are defined in the node's node definition file. There is a short explanation of the variable's effect on the node displayed when the user must specify the variable's value. Node variables can determine the following: produce and consume amounts of ports, the number of ports in a port array, the weight of the node (estimated execution time), and state structure initial contents.

Creating a Node

A node consists of its node definition, its name, its variable values, its location on the graph, and its port connections. To create a node, the first three parts must be specified. The port connections are later determined by the edges. To create a node, do the following:

1. **Select the Add Node function with the menus or *Ctrl-N*.** A file selection dialog displays a list of node definitions in the node definition directory. All node definitions are suffixed by .nd.
2. **Select a node definition file using the file selection dialog.** If the definition file is valid, a Node Variable Definition Dialog should appear prompting for the node name and the values of all node variables.

3. **Fill in the node's name and the values for all node variables in the node information dialog and select OK.** If name is unique and the values of all node variables are valid, the mouse pointer will become a pointing hand, and an image of the node will follow the mouse pointer.
4. **Move the new node to its correct location on the flowgraph display and click the *select* button to place the node.** If the location is valid, the new node will be placed at that location.

Creating Edges

An edge consists of a series of points with lines in between them. An edge must have at least two points and one line connecting them. The ultimate purpose of edges is to connect nodes, although during editing this is not required. To create a new edge do the following:

1. **Click the *edge-toggle* button or select the Add Edge function via the popup menu or *Ctrl-E.*** After this is selected, the cursor changes to a pencil pointer signifying add edge mode. The *edge-toggle* button will also toggle out of add edge mode, and will cancel any add edge operation in progress.
2. **Select a starting point for the edge by clicking the *select* mouse button.** The starting point will probably be the the inport or outport on the node, or the disconnected endpoint of an existing edge. Otherwise, this point will be the expected location of the inport or outport when the node is created, or the expected location of another edge's endpoint.
3. **Select other points in the edge path.** Clicking the *select* mouse button twice on the same point ends an edge. Clicking the select button on the inport or outport of a node also ends an edge (providing the edge doesn't connect two outports or two inports).
4. **To add more edges, repeat the previous two steps until done.** To exit add edge mode, simply click the *edge-toggle* button.

Selection

Selection is accomplished using the *select* button. All selections are displayed on the screen as a rectangle of cells in inverse video. The following explains the select operations:

- **Unselect**
  To unselect everything, click the *select* button inside an empty cell.
- **Single Selection**
  This is used to select a single node or edge intermediate point. Selecting a node is accomplished by clicking the mouse on a node. Selecting an edge intermediate point is accomplished by clicking on the cell containing that point. In both cases, the object selected will be highlighted in inverse video.
- **Range Selection**
  This is used to select a a range of nodes and edge intermediate points on the graph in a rectangular region. To make a range selection, press and hold down the *select* button at the cell location of one corner of the rectangle to select, and then drag the mouse pointer to the opposite corner of the rectangle while still holding down the *select* button. The rectangle selected will be shown on the display in inverse video. Now release the *select* button to complete the range selection.
- **Extending or Reducing a Selection**
  To enlarge or decrease the selected region, click or drag any corner of the rectangle with the *extend* button (left mouse button while holding down the shift key). The corner that follows the mouse will always be the closest one to the mouse pointer when the button is clicked.

Standard Cut/Copy to Clipboard

When using a selection with the **Cut** and **Copy** edit functions, the graph objects in a region are selected according to the following guidelines:

- Nodes are selected only if the complete node is within the selection region.
- Edges are selected in units of edge lines, where an edge line is a line connected to two edge points. Whenever both ends (edge points) of a line are inside the selection region, that edge line is selected.

Standard Move

When using a selection with the *move* edit functions, the graph objects in a region are selected according to the following guidelines:

- Nodes are selected only if the complete node is within the selection region.
- Any edge intermediate points are selected if they are inside the selection region.

Changing a Node

To change the name or variables of an existing node, do the following:

1. **Select a single node as explained above** (with either single selection or range selection).
2. **Select Change Node Vars from the edit menu or use the accelerator *Ctrl+V.*** If one and only one node is selected, a Node Variable Dialog should appear prompting for the node name and values of all node variables.
3. **Fill in the node's name and the values for all node variables in the node information dialog and select OK.** If name is unique and the values of all node variables are valid, the node will be changed and the screen will be updated.

NOTE

Changing the variables can change the number of inports or outports, so edges may be disconnected or connected automatically. Also, changing variables can effect the produce or consume amount of ports, causing or correcting mismatches across edges.

The Clipboard

The clipboard is a temporary area global to one XAssign application that can be used to easily transfer graph information from one graph to another or within a single graph. It holds the most recently cut or copied selection in the current XAssign application (in any one of its windows). The paste function copies the contents of the clipboard without modifying it into the selected window.

NOTE

Since the clipboard is global only to one XAssign application, all windows that wish to share data must be created with the New or Open functions from windows within the same XAssign session.

Edit Functions

Cut

The Cut edit function removes the objects in the selected area from the current graph. It removes all nodes and edges lines, and copies them into the XAssign clipboard. This may leave disconnected edges, but they can be connected later. Since objects removed with the cut function are copied to the clipboard, the cut function can be undone with paste function (below). The cut function is only allowed when there is a selected area.

Copy

The Copy edit function copies the selected area to the clipboard without modifying the current graph. It copies all the nodes and edges in the selected area to the XAssign clipboard without removing them from the graph. Copies of these objects can be pasted with the Paste function. The copy function is only allowed when there is a selected area.

Paste

The Paste edit function allows the contents of the clipboard to be placed on the current graph. After paste has been selected, you can use the mouse to place the contents of the clipboard (the objects will follow the mouse pointer). When the objects (nodes and edges) are in the correct location, click the left mouse button. Edges will automatically be connected to nodes and other edges if they share the same end point. If such connections are invalid or if this paste function would cause nodes or edges to overlap, a dialog will appear indicating that this paste location is not allowed. The paste function is only allowed when nothing is currently selected.

Dup

The Dup edit function duplicates the selected area on the current graph. It is similar to a copy function immediately followed by a paste, except that the objects duplicated are not copied to the clipboard, so the clipboard is unaffected by this edit function. The dup function is only allowed when there is a selected area.

Move

The Move edit function allows the current selection of nodes and edge points to be moved to a new location on the graph. Once move has been selected, the objects being moved can be placed in the same manner as the Paste function. The Move function is only allowed when there is a current selection.

Compiling a Flowgraph

Once you have created a complete flowgraph, you can compile it onto the host system or the iWarp array. The *Compile* pulldown menu contains three relevant selections:

Check Graph	Check the flowgraph for correctness
Host Compile	Compile onto the host system
iWarp Compile	Compile onto the iWarp array

The *Options* pulldown menu also contains two selections for passing options to the compiler.

Assign-Host	Edit options for host compilation
Assign-iWarp	Edit options for iWarp compilation

These menu selections are described in following sections.

Check Graph

Check Graph will verify that the flowgraph is valid. Both compile menu options invoke Check Graph before beginning compilation. When complete, Check Graph will display a window listing all of the errors found, or a message indicating that no errors were found. The following sections list some possible errors.

Possible Graph Errors

The following errors can occur when general rules regarding the flowgraph are violated.

- Multiple source nodes detected: *src2*, prev source *src1*
- Multiple sink nodes detected: *sink2*, original sink *sink1*
- Graph contains no source node
- Graph contains no sink node
- Graph is not acyclic
- Node *node* unreachable from (not connected to) source *src*

Possible Node Errors

The following errors can occur when nodes are not properly used.

- Node *node outport portname* produce amount= *amount* less than one
- Node *node* has disconnected Inports
- Node *node* has disconnected Outports
- Data Node *node* must have 2 or more input ports
- Data Node *node* must have 2 or more output ports

Possible Edge Errors

The following errors can occur when edges are not properly connected.

- Edge not connect to inport, OutNode *node*, OutPort *port*
- Edge not connect to outport, InNode *node*, InPort *port*
- Edge not connect to any port, starts at (*x1,y1*) and ends at (*x2,y2*)
- Edge Datatype Mismatch, InNode *node*, InPort *port*, OutNode=*node*, OutPort=*port*
- Edge Produce/Consume Mismatch, OutNode *node*, OutPort *port*, Produce=*amount*, InNode *node*, Outport *port*, Consume=*amount*.

Host and iWarp Compilation

The first step in these menu selections is to invoke the Check Graph routine described above. If any errors are detected, then compilation is not performed. Once a flowgraph passes Check Graph, the host or iWarp Compilations are then performed by invoking the assign driver. XAssign creates the input to the assign driver (started in pass 'a') by creating a *name.graph* file, where *name* is the name of the flowgraph. Once completed, the driver will leave the executables in the local directory. For iWarp compilation, a *go* script is also created.

After choosing one of these menu selections, XAssign opens a new window displaying the driver invocation, and any output that occurs in this process (such as debug or error messages). Once completed, the user must select the **OK** button to close the window. Any files created will be in the user's local directory.

Compiler Options

The *Assign-Host* and *Assign-iWarp* selections under Options will bring up an editable text window. Type any options to the driver, as if the driver were being invoked on a command line. Do not include the driver name or the name of the flowgraph. Spaces or carriage returns must be used to delimit the options. When finished, click on the **OK** button, or **CANCEL** to ignore any changes. **OK** will remember the options upon successive invocations of this menu selection.

The *Assign-iWarp* window is initialized with the following options:

- **-T** keep all intermediate files in the local directory
- **-Za** start the driver in pass 'a'
- **-##** print the invocation of each pass and its options

The *Assign-Host* window is initialized with the same options, plus the following option:

- **-host** compile for the host machine

For a complete description of the Assign compiler options, refer to Chapter 3.

Executing the Flowgraph

If compilation completes without any errors, you are ready to execute the flowgraph. For host compilation, simply type the name of the flowgraph on the command line (in same directory where XAssign was invoked). For iWarp compilation, type *go*, and a script will perform all the steps needed to load and run your executable on the iWarp array.

Menus

The XAssign menu system consists of a pulldown and popup menu containing the submenus: File, Edit, View, Compile, Options, and Help. The popup menu has the Edit menu in its main area for convenience, but allows access to the other items with submenus.

Pulldown

The pulldown menus are located on the menu bar at the top of the screen. The menu bar holds the pushbuttons for each pulldown menu. You can select the pulldown menu functions with the mouse or the keyboard.

To select a menu with the mouse, click the menu title with the mouse select button to pull down the menu, and then select the desired item. You can also select the pulldown menu from the keyboard with the F10 key. From there, you can select any menu item using the cursor keys to find the item, and the return or space key to select it.

Popups

Popup menus allow quick access to editing functions within the main display without the use of the keyboard, and access to all the other menu functions as well through the use of submenus. Popup menus are popped up by pressing the menu button (right button) on the mouse while in the drawing window.

The popup menu contains all the editing functions in the Edit menu, and the rest of the submenus are contained in submenus of the popup.

Mnemonics

Another way to select menu items is to use *mnemonics*. The underlined letters shown in the menu descriptions in the following sections are the *mnemonics* that allow keyboard selection. To select a pulldown menu, hold down the Meta key (a diamond on the suns) and press the mnemonic letter. Pressing the mnemonic letter while the menu is displayed selects the corresponding item.

Accelerators

The last selection method is the use of *accelerators*. Accelerators are key or key combinations that allow quick selection of functions without using the menu. Not every function has an accelerator, but the ones that do have the accelerator key or key combination listed following the description of the menu item in the following sections.

File

The file menu contains all functions performed on files. This includes:

New	Create a new flowgraph file. This will create a new window with an empty flowgraph unless this is a window with an empty flowgraph.
Open (Ctrl+O)	Open a previously existing flowgraph file.
Save (Ctrl+W)	Save the flowgraph file.
Save As	Save the flowgraph file under a new name.
Close (Escape)	Close the current flowgraph window.
Exit (Ctrl+X)	Exit the program and close all flowgraph windows.

Edit

The edit menu contains functions used to edit the flowgraph.

Change Node Vars (Ctrl+V)	Change the name and/or variable values for this node.
Add Node (Ctrl+N)	Add a new node to the flowgraph.
Add Edge (Ctrl+E)	Add a new edge to the flowgraph.
Cut (L10)	Remove the currently selected area and copy into the clipboard.
Copy (L6)	Copy the currently selected area into the clipboard (without removing it).

Dup (Ctrl+D)	Duplicate the currently selected area and allow it to be moved and placed. This is equivalent to selecting Copy and then Paste.
Paste (L8)	Paste the contents of the clipboard. The pasted contents can now be moved and placed with the mouse.
Move (Ctrl+M)	Move the currently selected area to a new location.

View

This menu allows changes in the view of the flowgraph.

Zoom In (Keypad +)	Zoom in the display toward the center of the current display.
Zoom Out (Keypad -)	Zoom out the display away from the center of the current display.
Grid Toggle (Keypad Enter)	Toggle the display grid on and off.

Compile

The compile menu allows you to perform utility functions such as compiling and running the graph.

Check Graph (Ctrl+G)	Check the graph for unconnected ports and edges and other things that would cause compiler errors.
iWarp Compile (Ctrl+i)	Compile the current flowgraph for the iWarp system.
Host Compile (Ctrl+H)	Compile the current flowgraph for the host system.

Options

Each of these items pops up a dialog box with compiler options.

Assign-Host	Popup a dialog for Assign Host compiler options.
Assign-iWarp	Popup a dialog for Assign iWarp compiler options.

Help

This menu gives you access to help on different aspects of XAssign. Selecting these causes a dialog to popup with the help text displayed. Scrollbars are used if needed for long help text.

General (Help)	Display help on window contents and areas.
On Viewing	Display help on viewing a flowgraph.
On Editing	Display help on editing a flowgraph.
On Compiling	Display help on compiling a flowgraph.
On Keys	Display help on keyboard shortcuts.
On Menus	Display help on the use of the popup/pulldown menus.
About	Display the version and copyright information.

Dialogs

There are several dialogs that appear when using XAssign. The following sections describe these dialogs.

Node Variable Definition Dialog

Figure C-2 illustrates a node variable definition dialog.

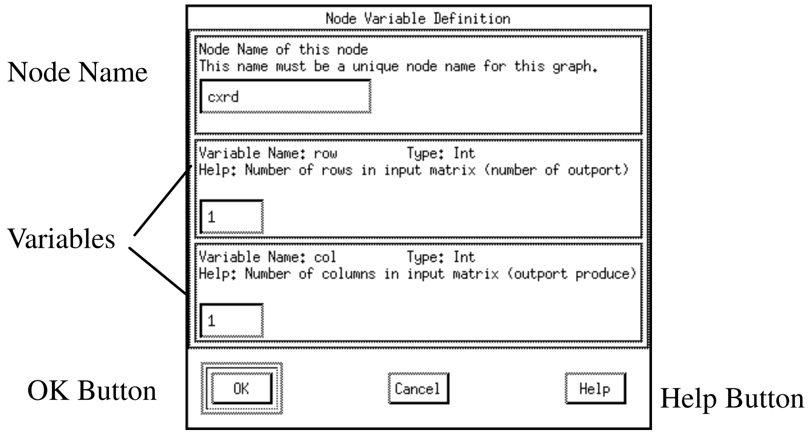

Figure C-2: Node Variable Definition Dialog

This dialog is displayed when you create a new node or change an existing node. The dialog displays an entry area for the node's name and for the value of each variable. To move between the text entry areas, use the TAB key or click the desired entry area with the mouse.

The node name field allows you to specify the unique name for a node. The default name is always the same as the node definition name. For large graphs, it is best to use easy to remember names. This way, it is easier to remember to which node error messages refer.

Together with the node name, the node variables define a node. The variables are all of integer type and a dialog with an error message will appear if the variables contain invalid values. The node variables are used for a number of purposes based on the node's definition. The following is a list of what the node variables can automatically configure:

- The node's state structure; all node variables should be in the state structure.
- The node's weight (weighted estimated execution time).
- The produce and consume amounts of each of the node's port.
- The number of ports in a vector of ports (for collect, distribute, and reduce nodes).

File Selection Dialogs

Figure C-3 illustrates a file selection dialog.

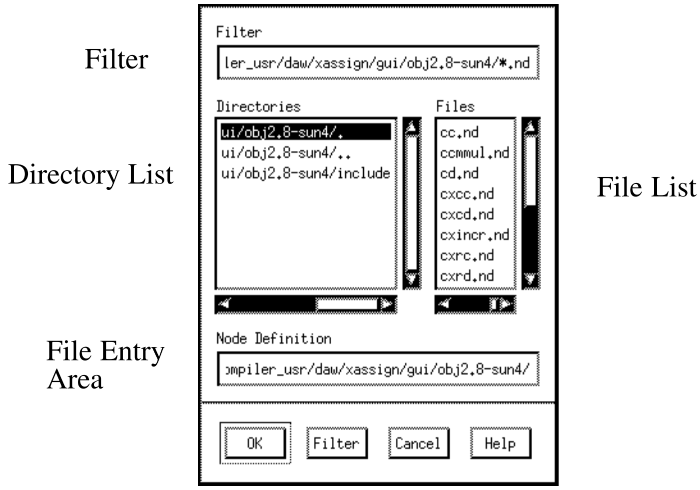

Figure C-3: File Selection Dialog for Node Definition

These dialogs are standard Motif dialogs that allow you to select files. You can use the file selection dialog to traverse directories, view files and subdirectories in them, and select files.

The File Selection Dialog has five areas:

Filter	The filter or directory mask text field. This is used for selectively displaying files in a directory. You enter the desired file mask here if you desire a different mask. This field will be set to the correct mask for the file type.
File List	The scrollable list of filenames. If a filename is selected by clicking it once, it will be put in the filename field. Clicking OK will then select the file. Quickly double clicking the filename will also select the file in one step.
Directory List	The scrollable list of subdirectories. Clicking a subdirectory will change to that directory and update the file list to display the contents of the directory with that mask.
File Entry Area	This field holds the name of the file currently selected. If you select OK, this filename is used.
OK, Filter, Cancel, and Help	The OK button returns the current contents of the filename input field. The Filter button updates the filename list with the current filter and directory. The cancel button cancels the current operation, and the help button displays help text.

Help Dialogs

Figure C-4 illustrates a help dialog.

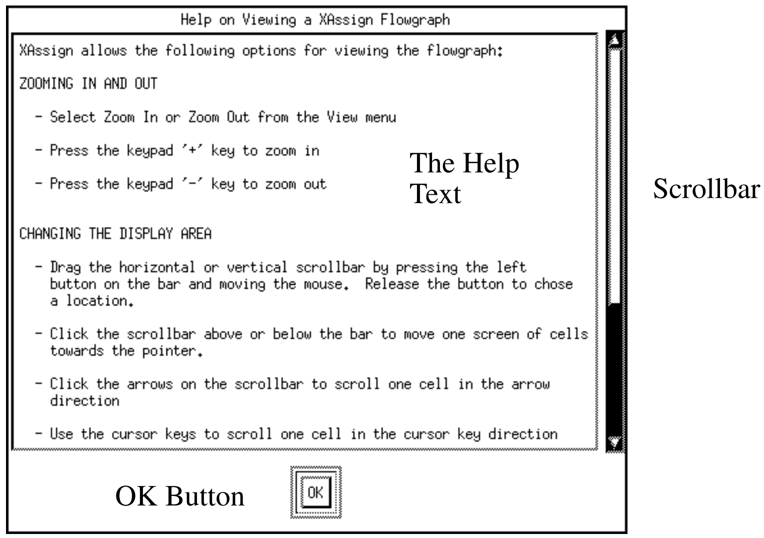

Figure C-4: Help Dialog

Every Help Dialog consists of a text display area and the OK button. If all of the help text does not fit in the text display area, scrollbars are provided to scroll through the text. The scrollbars operate in the same fashion as describe previously for the main window. Help text is provided from the help menus, from the file selection dialog, and from the node variable dialog.

Node Definition Files

A node definition file is a file containing information about a certain node type with the extension .nd. Any user adding an ASSIGN primitive must write a node definition file in order to use that node with XAssign.

The information in this file allows XAssign to do implement the node generation function. This information is required to allow XAssign to do the graph creation for a node.

Format

The following specifications describe the format and contents of each node definition file line header. The example text with each line header contains a running example of the definition of Fast Matrix Multiply. Blank lines are ignored, and any lines beginning with a “#” are assumed to be comment lines.

The following conventions are used in specifying the syntax of each definition line:

- **< >** angle brackets enclosing names indicate something the user must provide
- **[ ]** square brackets enclosing qualifiers indicate that qualifiers are optional
- **...** horizontal ellipsis indicate the the preceding optional item can appear one or more times in succession.

@name

```
@name <name> <type> [help]
```

Example:

```
@name fmmm comp help
This is the help for a Fast Matrix Multiply
.
```

The @name header should be on the first line of the node definition file. It defines the name and type of node, and the help text for the node. The name is the primitive name of the node. This name is used to construct the default names of the primitive function, the input function, and the output function by prefixing the name by prim_, in_, and out_, respectively. The type of node in <type> corresponds directly to the Assign node type definitions, and must be one of the following:

comp	A compute node. Most user defined nodes are of this type.
graph	A subgraph node. Subgraphs nodes are not implemented in XAssign, although you can do the equivalent using the copy and paste functions.
srd	Assign-defined sequential row distribute node.
scd	Assign-defined sequential column distribute node.
src	Assign-defined sequential row collect node.
scc	Assign-defined sequential column collect node.
sr	Assign-defined sequential reduce node.
prd	Assign-defined parallel row distribute node.
pcd	Assign-defined parallel column distribute node.
prc	Assign-defined parallel row collect node.
pcc	Assign-defined parallel column collect node.
pr	Assign-defined parallel reduce node.

The last optional parameter on the line is “help”. When the keyword help appears at the end of a line, help text is assumed to appear on the following lines and terminate with a line beginning with a period.

@var

```
@var <varname> [<type> [<default>]] [help]
```

Example:

```
@var mc int help
Number of rows in C (and A)
.
@var nc int help
Number of columns of C (and B)
.
@var na int help
Number of columns in A (rows of B)
.
```

The @var line(s) should appear just after the @name line and possible help text (all @var lines must appear before any expression). Each @var line defines a variable with its name, optional type, optional default value, and optionally ends with ‘‘help”. The only valid variable type is int, representing an integer variable. The type parameter is included for compatibility.

These variables are analogous to the parameters to the Assign Node Generation Function. The values of these variables are used in expressions to determine the following values during XAssign node creation:

- Weight of the node (estimated execution time)
- State structure initial contents (to pass information to the primitive)
- Port produce or consume amount.
- Number of ports in a port array (for Assign data nodes)

@inport, @outport

```
@inport <portname> <datatype> ”<amount>” [help]
@outport <name> <datatype> ”<amount>” [help]
```

Example:

```
@inport in0 float ”mc*nc” help
First Input Matrix (mc by nc)
.
@inport in1 float ”na*nc” help
Second Input Matrix (na by nc)
.
@outport out float ”mc*nc” help
Output matrix (mc by nc)
.
```

The @inport or @outport lines define one input or output port for a node with its name, data type, amount (consume for inport, produce for outport), and optionally ”help”. The amount is an integer expression enclosed in quotes. The integer expression follows C precedence rules (normal order of operations) with the following operators: **+**, **−**, *****, **/**, and ( ) (to change precedence). The datatype of these ports are the same as the Assign datatype #defines except are lower case:

int	Integer
float	Single precision floating point
double	Double precision floating point
complex_int	Complex Integer
complex	Complex single-precision floating point
complex_double	Complex double-precision floating point

@inportv, @outportv

```
@inportv <name> ”<numports>” <datatype> ”<amount>” [help]
@outportv <name> ”<numports>” <datatype> ”<amount>” [help]
```

Example: See node definition examples for Assign data nodes: cxrd & prred.

The @inportv or @outportv lines are currently only used for Assign data nodes. They define a vector or array of input output ports for a node. It is the same as the @inport or @outport lines above except it has the extra parameter ”<numports>” that is an expression for the number of ports in this port vector or array of ports.

@weight

```
@weight "<weight_expr>"
```

Example:

```
@weight "100*na*nc"
```

The @weight line sets the weight integer expression for the node type.

@inio, @outio

```
@inio [<inio_name>]
@outio [<outio_name>]
```

Example:

```
@inio
@outio
```

The @inio and @outio set the input and output function name, respectively, for a node. These lines are optional; if they are not included, the node is assumed to have no input or output function. If the name parameter is omitted, the default name is used. The default name is the string ‘‘in_” or ‘‘out_” followed by the node’s name.

@state

```
@state [<state_name>] "<state_expr>"
```

Example:

```
@state "mc, nc, na"
```

The @state line sets up the contents and name of the state structure. If the state name is specified it is used as the name of the state structure; otherwise, the default name is used (”state_” followed by the node’s name). The state expression contains the parameters used to initialize the state structure as expressions separated by commas.

@obj, @inc

```
@obj [<obj1> ...]
@inc [<inc1> ...]
```

Example:

```
@obj fmmm.o
@inc fmmm.h
```

The @obj and @inc lines define all the object and header files, respectively, that are used by this node type. Up to five filenames are allowed per line, and multiple lines of either are allowed.

Example Definitions

Compute Node: Power

This is an example of a compute or atom node, the most common node definition type you will write. Compute nodes are used mainly for computation and have fixed input and output ports and data types, although the produce and consume amounts on those ports usually change depending on the variables.

```
#
# Name: POWER
# Type: Compute Node
#
@name power comp help
19 August 1992

NAME
     POWER - Calculate the power of a Complex number vector

PURPOSE
     Accepts a vector of complex numbers and outputs a vector of
floating
     point numbers holding the power of the complex numbers.

STATE STRUCTURE VARIABLES
     size       Size of input and output vectors

INPUT PORT
     in         Complex Input vector of size "size"

OUTPUT PORT
     out        Float Output vector of size "size"

DESCRIPTION
     POWER calculates the power of each complex number in the input
vector
     "in" which is simply the square root of the sum of its real and
complex
     components squared.

EXAMPLE
     Input:  size = 4
             in:  { (3.0, 4.0), (-5.0, 12.0) }

     Output:
             out: { 5.0, 13.0 }
.
#no "@obj power.o" because of standard library
#
#Use include file power.h
@inc power.h
#
```

```
#This node has an input routine (named in_power)
@inio
#
#This node has an output routine (named out_power)
@outio
#
#Defines variable size to be of type int and have help
@var size int help
Size of input and output vectors.
.
#
#The weight of this node is 500*size, where size is the value of
#variable size
@weight ”500*size”
#
#The state structure contains only the value of the size variable
@state ”size”
#
#Define inport ”in” accepting complex data of size ”size” and help
@inport in complex ”size” help
Input vector to be converted to power vector
.
#
#Define outport ”out” receiving float data of size ”size” and help
@outport out float ”size” help
Output vector contain power elements
.
```

Reduce Node: Parallel Row Reduce

Reduce nodes are useful for reducing the input from a number of input ports into one output port. In order to handle this, reduce nodes have a dynamic number in input ports, and one output port. See the `@inportv` definition line to see how to set up the vector of input ports.

```
#
# Name: PRRED
# Type: Parallel Row Reduce
#
@name prred pr help
14 July 1992

NAME
     PRRED – Parallel plus row–reduce of ROW vectors of COL–element

PURPOSE
     To sum ROW vectors of size COL of float.  Output a vector of size
COL
     containing this vector sum.

STATE STRUCTURE VARIABLES
     row        Number of input vectors to reduce.

     col        Size of each input vector and the output vector.
```

```
INPUT PORTS
     in0, in1, .. in<col-1>
                 Each is a input vector of size "col" of float
                 There are a total of "row" input ports.

OUTPUT PORT
     out         Vector of size "col" of float

EXAMPLE
     Input:  row= 2, col= 3
             in0:  {  1.0,  2.0 }
             in1:  {  3.0,  4.0 }
             in2:  {  5.0,  6.0 }

     Output:
             out:  {  3.0, 7.0, 11.0 }
.
#no "@obj prred.o" because there is a standard library
#
#Use include file prred.h
@inc prred.h
#
#Node has input routine (named in_prred)
@inio
#
#Node has output routine (named out_prred)
@outio
#
#Defines variable row to be of type int, have default 1, and have help
@var row int 2 help
Number of input vectors or rows in input matrix
.
#
#Defines variable col to be of type int, have default 1, and have help
@var col int 1 help
Size of each input vector and the output vector or columns in matrix
.
#
#Define the weight
@weight "100*row*col"
#
#The state structure only contains the col variable
@state "col"
#
#Vector of inports name in1, in2,...
#There are "row" of them, of type float, size "col", with help
@inportv in "row" float "col" help
Input vectors of size "col" of floats (There are "row" of them).
.
#
```

```
#Outport named out, type float, size ”row”, with help
@outport out float ”col” help
Output vector of size ”col” of floats
.
```

Distribute Data Node: Complex Row Distribute

Distribute data nodes are used to distribute data from one input port to a variable number of output ports. See the `@outportv` definition line to see how to set up the vector of output ports. There is no example of a collect node included, but they are analogous to these nodes, except they collect data from a variable number of input ports and distribute it to one output port.

```
#
# Name: CXRD
# Type: Complex Row Distribute
#
@name cxrd prd help
14 July 1992

NAME
     CXRD - Complex Row Distribute

PURPOSE
     To distribute an input matrix into multiple output vectors where
     each output vector contains one row of the matrix.

STATE STRUCTURE VARIABLES
     row        Number of rows in output matrix and also
                the number or output ports in this node.

     col        Number of columns in output matrix and also
                the amount of each output port

INPUT PORT
     in         Complex Input matrix of total size row*col

OUTPUT PORTS
     out0, out1, .. out<row-1>
                Each is a output vector of size row of complex
                There are a total of ”row” input ports.

EXAMPLE
     Input:  row = 1, col = 3
             in:  { (-5.0, -3.0),
                    ( 1.0,  2.0),
                    (-4.0,  3.0)  }

     Output:
             out0: { (-5.0, -3.0) }
             out1: { ( 1.0,  2.0) }
             out2: { (-4.0,  3.0) }
```

```
.
#@obj cxrd.o not needed because we’re using a library
#
#Use include file cxrd.h
@inc cxrd.h
#
#Has input routine (in_cxrd) and output routine (out_cxrd)
@inio
@outio
#
#Defines variable row to be of type int, have default 2, and have help
@var row int 2 help
Number of rows in input matrix (number of outports)
.
#
#Defines variable col to be of type int, have default 1, and have help
@var col int 1 help
Number of columns in input matrix (outport produce)
.
#
#Define the weight
@weight ”100*row*col”
#
#The state structure only contains the col variable, and 0
@state ”col,0”
#
#Inport named in, type complex, size ”row*col”, with help
@inport in complex ”row*col” help
Input matrix with each input port as one row.
.
#
#Vector of outports name out1, out2,...
#There are ”row” of them, of type float, size ”col”, with help
@outportv out ”row” complex ”col” help
Vector of output vectors to be collected into a matrix.
.
```

Customizing XAssign

Setting Motif Key Bindings

The Motif key bindings can be set either in the *.motifbind* file in your root directory, or by setting the resource **defaultVirtualBindings*. The preferred method is to setup your *.motifbind* file because it is global to all Motif applications, however, this only works with the Motif Window Manager (mwm). Sample files for different machines are found in the Motif distribution, and below is an example file. Note that the backspace and delete keys are switched. Be sure to include bindings for all keys shown.

```
osfCancel        :                <Key>Escape
osfLeft          :                <Key>Left
osfUp            :                <Key>Up
osfRight         :                <Key>Right
osfDown          :                <Key>Down
osfEndLine       :                <Key>F33
osfBeginLine     :                <Key>F27
osfPageUp        :                <Key>F29
osfPageDown      :                <Key>F35
osfBackSpace     :                <Key>Delete
osfDelete        :                <Key>BackSpace
osfInsert        :                <Key>Insert
osfAddMode       :Shift           <Key>F8
osfHelp          :                <Key>Help
osfMenu          :                <Key>F4
osfMenuBar       :                <Key>F10
osfCopy          :                <Key>F16
osfCut           :                <Key>F20
osfPaste         :                <Key>F18
osfUndo          :                <Key>F14
```

To set the resource *defaultVirtualBindings, add the following to your defaults file:

```
*defaultVirtualBindings: \
    osfCancel    :               <Key>Escape\n\
    osfLeft      :               <Key>Left\n\
etc...all keys
```

Be sure to include the \n\ at the end of each line except the last one.

Setting up the FontList

To allow XAssign to use a new font, add the following to your defaults file:

```
*fontlist: fixed
```

The font *fixed* can be substituted for any valid font on your system.

Setting the default size of the Display Area

To set the default display area size, add the following to your defaults file. This example sets the width and height to 480, a fairly large display.

```
*draw.width:  480
*draw.height: 480
```

Tutorial Example

This tutorial shows how to create the ABI example shown previously in the manual using XAssign.

Is Everything Ready?

Change directory to the XAssign tutorial directory. This directory should contain at least the following files in order for this tutorial to run:

The Node Definition files	cxcc.nd, cxincr.nd, cxrd.nd, power.nd, and prred.nd
The header files	cxcc.h, cxincr.h, cxrd.h, power.h, and prred.h
The libraries	libhost.a and libiw.a

If the libraries and header files are not present, but the .prim files are present, use the **make -f prim_make** command to set up the libraries and header files.

XAssign should also be installed on your system, and your path environment variable should also be set up so XAssign's install directory is included.

Startup

To start XAssign (and this tutorial), enter the following command at the command line prompt (in your XAssign directory):

```
xassign &
```

This runs XAssign as a background process. This is convenient because XAssign does not send output to the terminal window unless it fails on startup. Figure C-5 shows an XAssign window at startup.

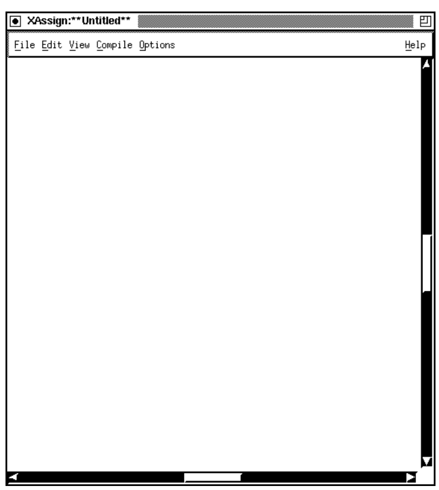

Figure C-5: XAssign Window at Startup

Creating the rcfan graph

This *rcfan* graph will take a matrix of complex numbers as input and do the following:

- Divide that matrix into rows with a Complex Row Distribute node.
- Increment each complex number with a Complex Increment node.
- Collect the result into a matrix with each row being a column in the new matrix (the transpose) with a Complex Column Distribute node.

The first step in creating a flowgraph is creating nodes. The first node needed is a Complex Row Distribute node. To create this node, select the **Add Node** function from the Popup or Pulldown menu, or use its accelerator *Ctrl+N*. Since this is an editing function, use the popup menu since you will use the mouse after the function is selected. To bring up the popup menu, press the right mouse button. Select **Add Node**, either by clicking it with the right button, or by holding down the right button, and releasing the button on top of **Add Node**.

Now a file selection dialog with the Node Definitions listed should appear as shown in Figure C-6.

Filter

Directory List

File List

File Entry Area

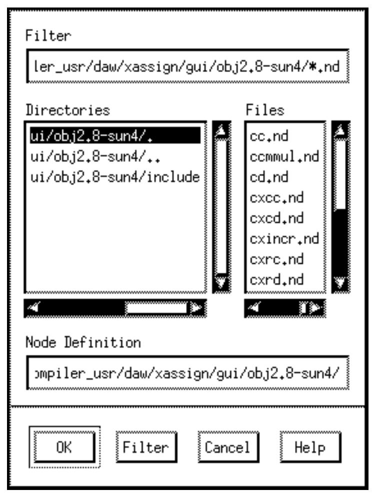

Figure C-6: File Selection Dialog for Node Definitions

The Node Definition File for this type of node is **cxrd.nd**, so look through the file list for that file and click the file, or type the filename in the text entry area under Node Definition. Now click OK to select the file. Another way to select the file is to quickly double click the filename in the file list.

A node variable definition dialog for the cxrd.nd should appear for complex row distribute as shown in Figure C-7.

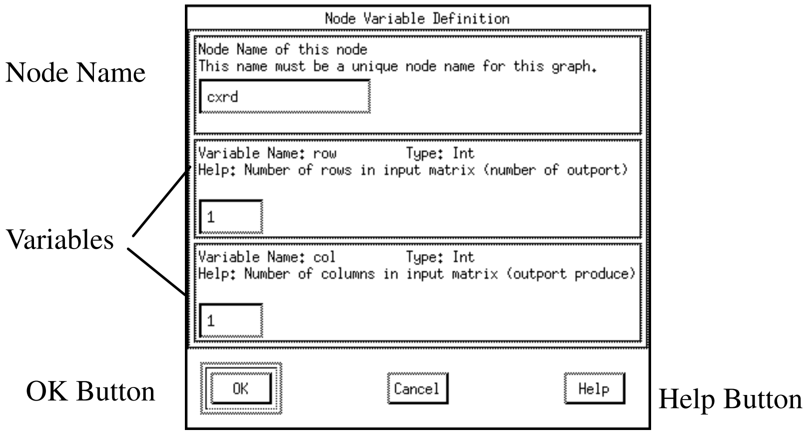

Figure C-7: Node Variable Dialog

Now, enter 5 for the *row* variable, and 4 for the *col* variable. These numbers were picked arbitrarily for this example. If you would like more details on this node, click the help button. A help dialog explaining this node will appear. Click the OK button to accept these values.

The image of the node will follow the mouse. Since this will be the source node, move the mouse to the left center of the screen, and press the left mouse button the place the node there, as shown in Figure C-8.

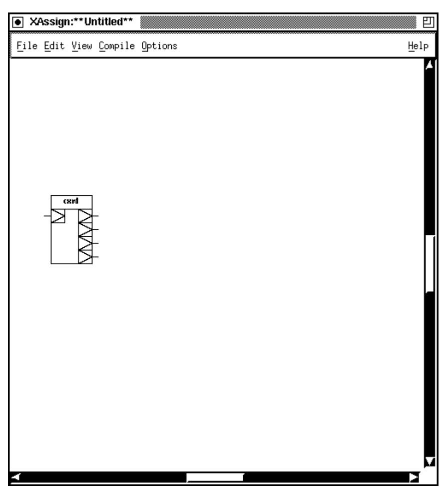

Figure C-8: Placing the Row Distribute Node

The next step is to create some complex increment nodes to connect to the source node. To do this, select **Add Node** again. Then, in the file selection dialog, select **cxincr.nd**.

A Node Variable Definition Dialog will appear for the complex increment node. Change the variable *size* to 5. The size must be five because the row distribute node has a row size of five, and each output vector from the row distribute is one row from the original matrix. Click OK to accept these values.

Move the mouse to the top center of the window and place the node there by clicking the left button. This node will be the first row vector outport.

Since we need a total of four of these type of nodes with the same size variable, the fastest way to generate the other three is with the **Dup** editing function. To do this, first select the cxincr node we already created. Do this by simple clicking the left mouse button inside the node. The display should now look like figure C-9.

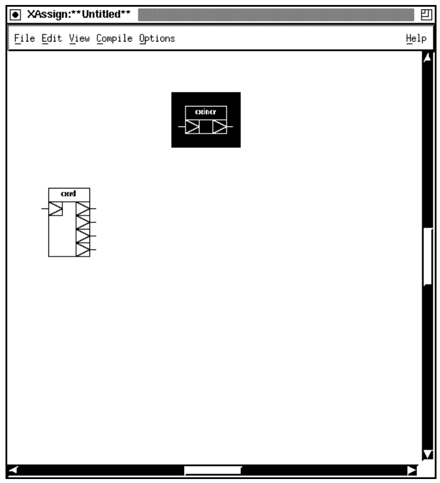

Figure C-9: Selecting the cxincr node for a Dup

Now that the node is selected, select the **Dup** function, either from the Edit pulldown, or from the popup as described above. A copy of the selected node will follow the mouse. Move the mouse to a few cells below the original node and click the left button to place it there. Note that a new unique name is automatically generated for the node, using the base name.

Repeat the above procedure two more times to create the total of four nodes, as shown in Figure C-10.

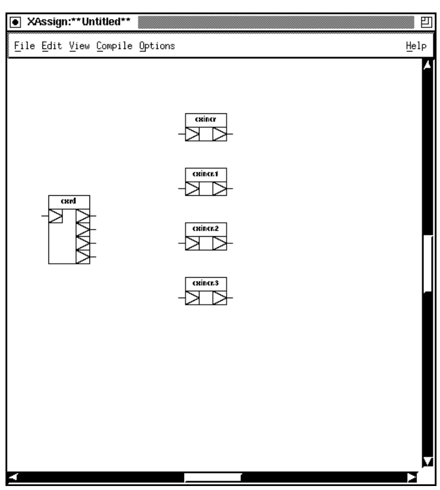

Figure C-10: Four cxincr nodes successfully created

If the nodes are not exactly where you want them, they can be moved individually or as a group. Move these four nodes as an exercise even if they are already where you want them.

To do this, you must select the rectangle containing the nodes. Note that all cells adjacent to the main rectangle of a node must be selected to select a node. This is accomplished by doing the following:

1. Move the mouse to a corner of the rectangle to be selected.
2. Press and hold down the left mouse button.
3. Move the mouse to the opposite corner of the rectangle to be selected (the rectangle is highlighted in inverse video).
4. Release the left button.

All of the cxincr nodes should now be selected, as shown in Figure C-11.

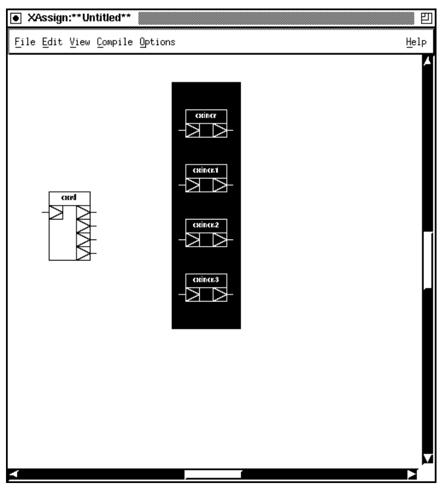

Figure C-11: All cxincr nodes selected for a Move

Select the **Move** editing function with the pulldown, popup, or *Ctrl+M*. All four nodes should now follow the mouse. Move the nodes to the desired location and then click the left button to place the nodes.

Now create the last node, a complex column collect, and specify 4 for the *row* value, and 5 for the *column* value. Notice that the values of the row and column variables are opposite what they were for the complex row distribute node. The vectors that were originally considered rows are now columns because the matrix is being transposed. All the nodes are now created, as shown in Figure C-12.

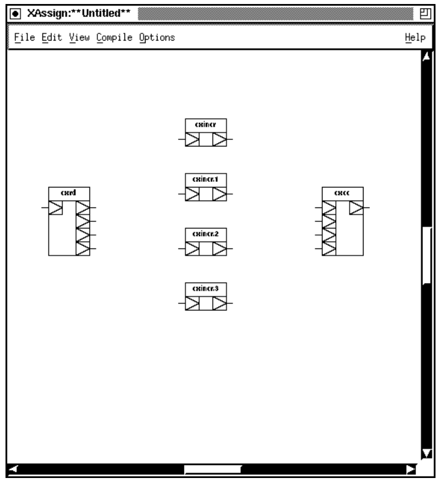

Figure C-12: All node created

The next step is to create the edges. To add edges, select **Add Edge** from the popup or pulldown or press the middle mouse button. Now the mouse pointer should look like a pencil. To connect two ports, simply click the left mouse button on one of the ports, move the mouse to the other port (the edge line should follow the mouse), and click the mouse button on the other port. If you would like the edge to have an intermediate point, click the mouse on that point on the way to the other port. If you would like to end an edge without connecting it to a port, simply click twice on the end point. To continue an edge that is not connected to a port, just click on the end point and proceed with adding edge points or connecting to a port.

Now, as instructed previously, connect the top cxincr node to the top outport of the cxrd node and top inport of the cxcc node, and so on until all the cxincr nodes are connected as shown in Figure C-13.

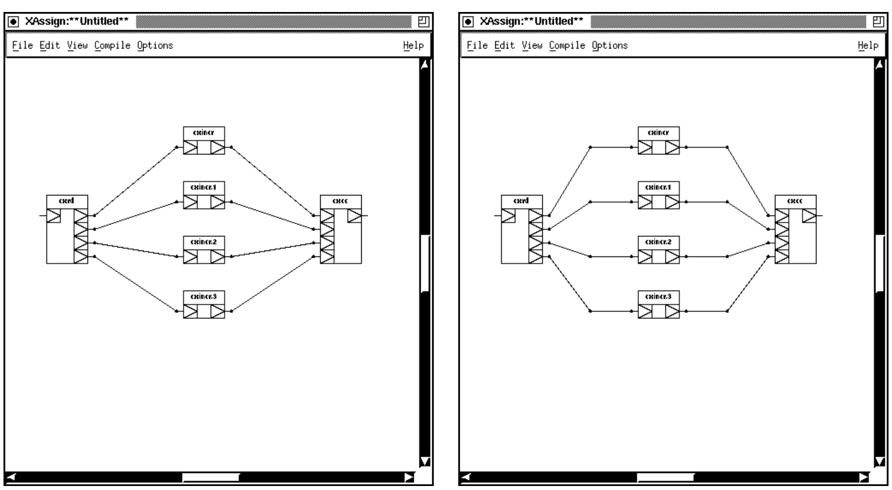

Figure C-13: Two possible graphs with the same port connections

Now you have created an XAssign graph. To make sure the graph is valid, select **Check Graph** from the Compile menu of the pulldown or popup menu. If everything is valid, a dialog will display telling you that the graph is valid.

To illustrate the **Change Node Vars** function as well as the **Check Graph** function, select the cxrd node by clicking it with the left button, and select **Change Node Vars** from the popup or pulldown Edit menu. Change the *row* variable from 4 to 6 and click OK to accept the change. Now the selected node should have two more outports. Now select the **Check Graph** function. A dialog should appears saying:

```
Error: Node "cxrd" has disconnected Outports
```

Now, as above, change the *row* variable from 6 to 2, and select **Check Graph** again. This time more errors will be printed. Your errors should be similar to these:

```
Error: Edge not connect to outport, InNode "cxincr.3", InPort "in"
Error: Edge not connect to outport, InNode "cxincr.2", InPort "in"
Error: Node "cxincr.3" unreachable from (not connected to) source "cxrd"
Error: Node "cxincr.2" unreachable from (not connected to) source "cxrd"
```

The first two errors are self-explanatory. The last two errors are generated because all nodes should be reachable from the source node, or else they are not part of the flowgraph.

Now fix the node by changing the *row* variable from 2 back to 4. Select **Check Graph** again to verify that everything is valid. Now select **Compile Host** from the Compile menu or use *Ctrl+H*. Your graph should now compile successfully in a compile window, as shown in Figure C-14.

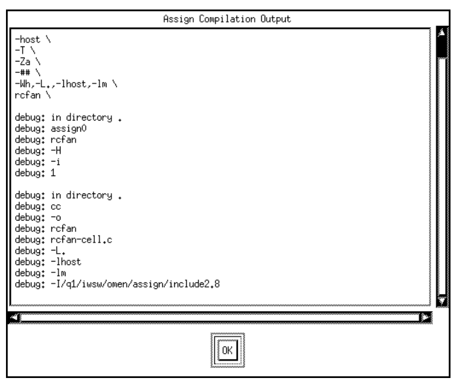

Figure C-14: Compilation Window

Compiling an Assign program for the host generates an executable with the graph's base name. So select the terminal window from which you invoked XAssign, type **rcfan** and press return. This runs the host version of the flowgraph program. The input automatically generated by the complex row collect node's (cxrc's) input function looks like this:

```
ROW 0: (   0.0,   0.0) (   1.0,   1.0)...(   3.0,   3.0) (   4.0,   4.0)
ROW 1: ( 100.0, 100.0) ( 101.0, 101.0)...( 103.0, 103.0) ( 104.0, 104.0)
ROW 2: ( 200.0, 200.0) ( 201.0, 201.0)...( 203.0, 203.0) ( 204.0, 204.0)
ROW 3: ( 300.0, 300.0) ( 301.0, 301.0)...( 303.0, 303.0) ( 304.0, 304.0)
```

The output is the input plus one transposed, so verify that the output of the graph in the terminal window looks like this:

```
ROW 0: (   1.0,   1.0) ( 101.0, 101.0)...( 201.0, 201.0) ( 301.0, 301.0)
ROW 1: (   2.0,   2.0) ( 102.0, 102.0)...( 202.0, 202.0) ( 302.0, 302.0)
ROW 2: (   3.0,   3.0) ( 103.0, 103.0)...( 203.0, 203.0) ( 303.0, 303.0)
ROW 3: (   4.0,   4.0) ( 104.0, 104.0)...( 204.0, 204.0) ( 304.0, 304.0)
ROW 4: (   5.0,   5.0) ( 105.0, 105.0)...( 205.0, 205.0) ( 305.0, 305.0)
```

Now that you have created and tested the *rcfan* graph, select **Save** or **Save As** from the File menu. These are equivalent when the graph does not have a name, but **Save** will save the graph immediately with the current name if one does exist. Enter the name *rcfan.xgr* for the graph name. Always use the extension *.xgr* when specifying graph names, so XAssign will automatically list your graph with the other XAssign graphs.

Creating the prfan graph

This graph takes a matrix of complex numbers as input and does the following:

- Divide the matrix into rows with a complex row distribute node.
- Convert each complex number into a floating point number representing its power (the square root of the sum of the squares of its real and imaginary parts) with a power node.
- Sum all the row vectors to output one result vector with a parallel reduce node.

Select **New** from the File menu. A new window will appear. Create a complex row distribute node: cxrd.nd. Set its *row* variable to 5 and its *col* variable to 4, and place the node on the left center of the screen. Now create five power nodes, *power.nd*, to connect to the output ports. Each should have its *size* variable set to 4. Place them in the center of the screen. Finally, create a parallel summing reduce node, prred.nd, with is *row* and *col* variables the same as the distribute node. Connect all the nodes together with edges as shown in Figure C-15.

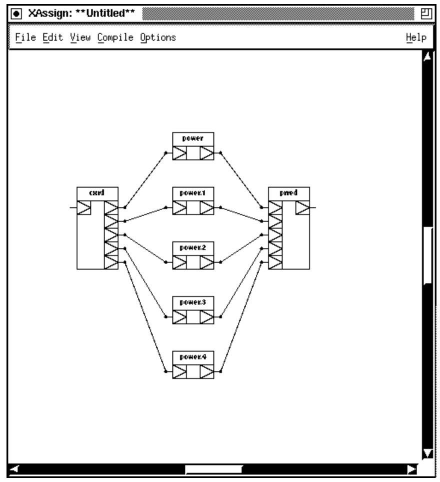

Figure C-15: prfan Graph

Check and compile the graph with the **Compile Host** option. Run the program in your terminal window and you should get the following output:

```
1414.213623
1421.284668
1428.355713
1435.426758
```

Since the input follows the same pattern as the input above, the first output should be the sum of the powers of the complex numbers: (0,0), (100,100), (200,200), (300,300), (400,400). If you work it out, this is just the sum (0+100+200+300+400)*sqrt(2) = 1000*sqrt(2) which is about 1414. The other values work out similarly.

Now use **Save** or **Save As** to save your graph as *prfan.xgr*.

Putting it all together

The final graph is a single graph with the output of the first graph as the input of the second graph. First, create another graph with the **New** function. The next step is to paste the contents of the other two windows into this one. If needed, use the **Open** function to retrieve the saved rcfan and prfan graphs. First, select the entire contents of of the rcfan graph by pressing the mouse button on one corner of the rectangle and dragging the mouse to the other corner so all nodes and edges in the graph are contained in the rectangle, as shown in Figure C-16.

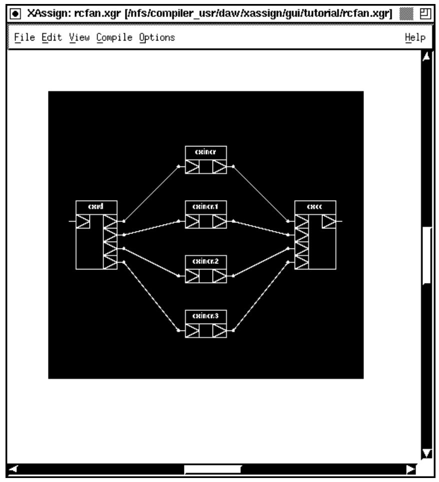

Figure C-16: rcfan graph selected

Now select the **Copy** edit function to copy this area to the clipboard.

Paste the rcfan graph from the clipboard into the new window using the **Paste** function, and place it on the left side of the screen. You may want to select **Zoom Out** from the view menu to allow room for the other graph in the display area. Use the same method to copy the prfan graph to the right side of the screen. Then connect the two graphs with an edge using the **Add Edge** function. The final graph should look like Figure C-17.

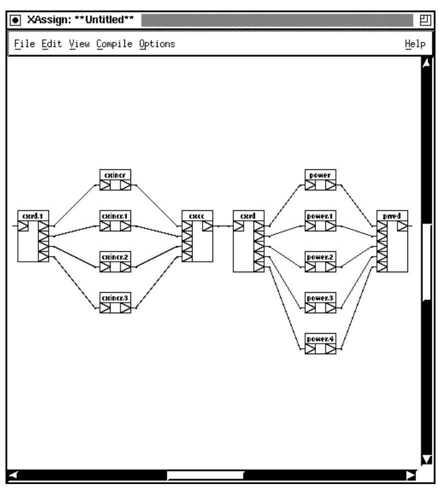

Figure C-17: Complete graph example

Select **Compile iWarp** from the compile menu to compile the graph. After the graph successfully compiles, select the window from which you started XAssign, and type *go* to run the compiled graph on the iWarp array. Type the unix command **cat stdout*** to see the results:

```
21.213203
728.320007
1435.426758
2142.533447
```

Once again, these results are as expected because the input to the second graph is the output of the first, so the first number is sum of the powers of (1,1), (2,2), (3,3), (4,4), (5,5) which is 15*sqrt(2)=21.21.

You have now completed the process of creating, compiling, running, and testing a flowgraph with XAssign.