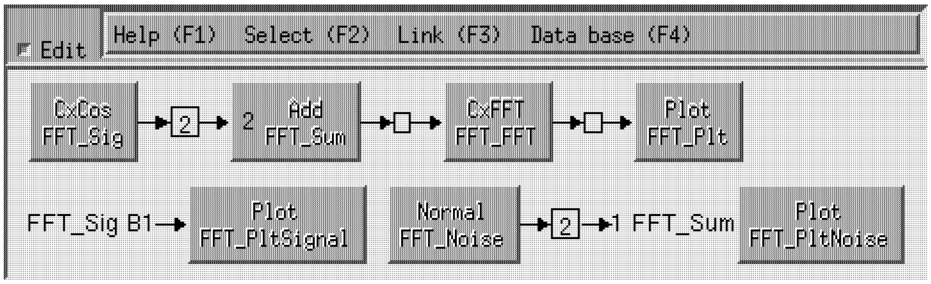
ObjectProDSP Overview and Tutorial

Paul P. Budnik Jr. Phd.
Internet: support@MTNMATH.COM

September 1994

'.dvi' file created September 16, 1994



Mountain
Math
Software
P. O. Box 2124, Saratoga, CA 95070
Fax or voice (408) 353-3989

Published by Mountain Math Software, P. O. Box 2124, Saratoga, CA 95070.

Copyright © 1994 by Mountain Math Software. All rights reserved.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the sections entitled "GNU General Public License" and "Licensing" are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one, and provided the derived work is clearly identified as a derived work and not solely the creation of either the orignal authors or the authors of the derived work.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the sections entitled "GNU General Public License" and "Licensing", and this permission notice, may be included in translations approved by Mountain Math Software instead of in the original English. Translations of the section entitled "GNU General Public License" must also be approved by the Free Software Foundation which owns the copyright to that text.

Licensing

ObjectProDSP^{TM} is licensed for free use and distribution under version 2 of the GNU General Public License. See Appendix C for the full text of this license. There is absolutely no warranty for ObjectProDSP under this license. ObjectProDSP is a trademark of Mountain Math Software.

You are free to use and distribute ObjectProDSP under the terms of version 2 of the GNU General Public License. Please note that *none* of the ObjectProDSP system is licensed for use under the GNU *Library* General Public License. The Gnu General Public License allows you to distribute executables or librarys linked with or created by ObjectProDSP *only* if you make *all* the *source* code used to create the librarys or executables (other than standard librarys that are part of a compiler or operating system) freely available. Please read the license in Appendix C for the full legal explanation of these conditions.

Mountain Math Software plans to offer, for a fee, a commercial version that will allow you to distribute executables generated with ObjectProDSP under standard commercial terms.

If you wish to extend ObjectProDSP you can distribute your code with ObjectProDSP under the terms of the GNU General Public License. If you include an appropriate copyright notice in your name for your upgrades then no one, including Mountain Math Software, will be able to distribute your code under any terms other than the GNU General Public License without your permission.

If you find ObjectProDSP useful in a commercial environment you are asked to consider purchasing a support contract. This is not shareware and you are under no obligation to do so but you will gain aceess to direct support from Mountain Math Software and you will make a contribution to the continued success of ObjectProDSP and thus to any of your endeavors that benefit from it.

If you are interrested in a custom port of ObjectProDSP to directly support your company's DSP development board or processor please contact us.

Mountain Math Software
P. O. Box 2124
Saratoga, CA 95070
Internet: support@MTNMATH.COM
Fax or voice (408) 353-3989

Documentation

- *ObjectProDSP Overview and Tutorial* This is the document you are reading. This gives a general description of ObjectProDSP's purpose and function. It includes several tutorial examples. There are appendices on the DSP node and class library and Mountain Math Software.
- *ObjectProDSP User's Reference* This describes the user interface and DSP++, a C++ based language for DSP. (You do not need to known DSP++ or C++ to use ObjectProDSP. DSP++ statements are generated for you when you graphically enter a network or execute menu data base commands.) This document includes a reference manual for the menu data base. Appendixes contain a synopsis of menu data base commands and a general index.
- *ObjectProDSP Library Reference* This gives a detailed description of ObjectProDSP interactive objects including DSP processing nodes.
- *ObjectProDSP Developer's Reference* This tells how to write DSP processing nodes and add them to ObjectProDSP. It describes ObjectPro++^{TM}, an extended C++ language for defining interactive objects for DSP or other applications. It explains how to modify the part of the menu data base that does not come from interactive object definitions in ObjectPro++. It describes how to update the ObjectProDSP manuals to include your new nodes and objects. Information about these objects is extracted from your definitions by ObjectPro++ and added to the manuals.

Contents

Licensing iii

Documentation v

List of figures ix

1 Purpose 1

2 Initial windows 4

3 Spectral analysis example 6

4 FIR filter frequency response 14

5 Multi-stage FIR filter 20

6 Constructing and editing DSP networks 26

Appendix A ObjectProDSP library

Appendix B Mountain Math Software

Appendix C GNU GENERAL PUBLIC LICENSE

Index

List of Figures

1 CASE tool for DSP analysis design and implementation 2

2 Initial windows 5

3 Network from spectral analysis example 6

4 FFT plot from spectral analysis example 7

5 Expanded spectral plot 8

6 Power spectral plot 9

7 Decibel spectral plot 10

8 Program for spectral analysis example 11

9 Menu for FFT_Sig object 12

10 Description of FFT_Sig object 13

11 Network to compute FIR filter response 14

12 Complex plot of FIR filter frequency response 15

13 Power plot of FIR filter frequency response 16

14 Decibel plot of FIR filter frequency response 17

15 Menu for Fir_resp_cximp object 17

16 Description of Fir_resp_cximp object 18

17 Program for FIR filter response 19

18 Network for multiple stage filter 20

19 Spectral plot of multi-stage filter input 21

20 Spectral plot of multi-stage filter output 22

21 Menu for Fir_net object 23

22 Target code generation with large buffers 24
23 Editing a CxFir node into FFT_Net. 26
24 Selecting FFT_Sum to connect to. 27
25 Completed edit of FFT_Net. 28
26 Menu to execute edited network. 29
27 New complex plot from edited network. 29
28 Decibel power view of plot. 30

1 Purpose

ObjectProDSP^{TM} provides a high level environment for design and implementation of complex Digital Signal Processing (DSP) systems. You can interactively define a DSP network, create test data and display output in graphical or numeric form. You may change processing parameters or edit the network topology interactively. Network consistency checks and analysis of sample rates and sample times are performed on data streams. The arithmetic of supported targets is simulated. Stand alone code is generated at the touch of a button for real time execution.

Any target processor with an Ansi C compiler can be supported. All the standard DSP objects can be ported. It may be desirable to code portions of some in assembly language for optimum performance. The kernel of a DSP object can be coded in C++, C or assembly language.

Choosing software development tools involves involves tradeoffs such as:

- Efficiency versus ease of programming (for example assembly language versus a high level language).
- General purpose low level tools versus application specific high level tools.
- Interactive tools for rapid development versus compiler based tools for efficient code.

ObjectProDSP minimizes these tradeoffs. You can take on complex DSP development applications with confidence. This is achieved by applying the techniques of object oriented programming to DSP in an integrated package. The X-windows/InterViews graphical user interface makes it easy to understand and manipulate these objects.

C++ provides the framework for this structure. A DSP language, DSP++ is defined within C++ by defining DSP classes. Objects are saved as statements in this language. Programs in this language can be written directly or can be generated automatically by graphically defining and editing DSP networks.

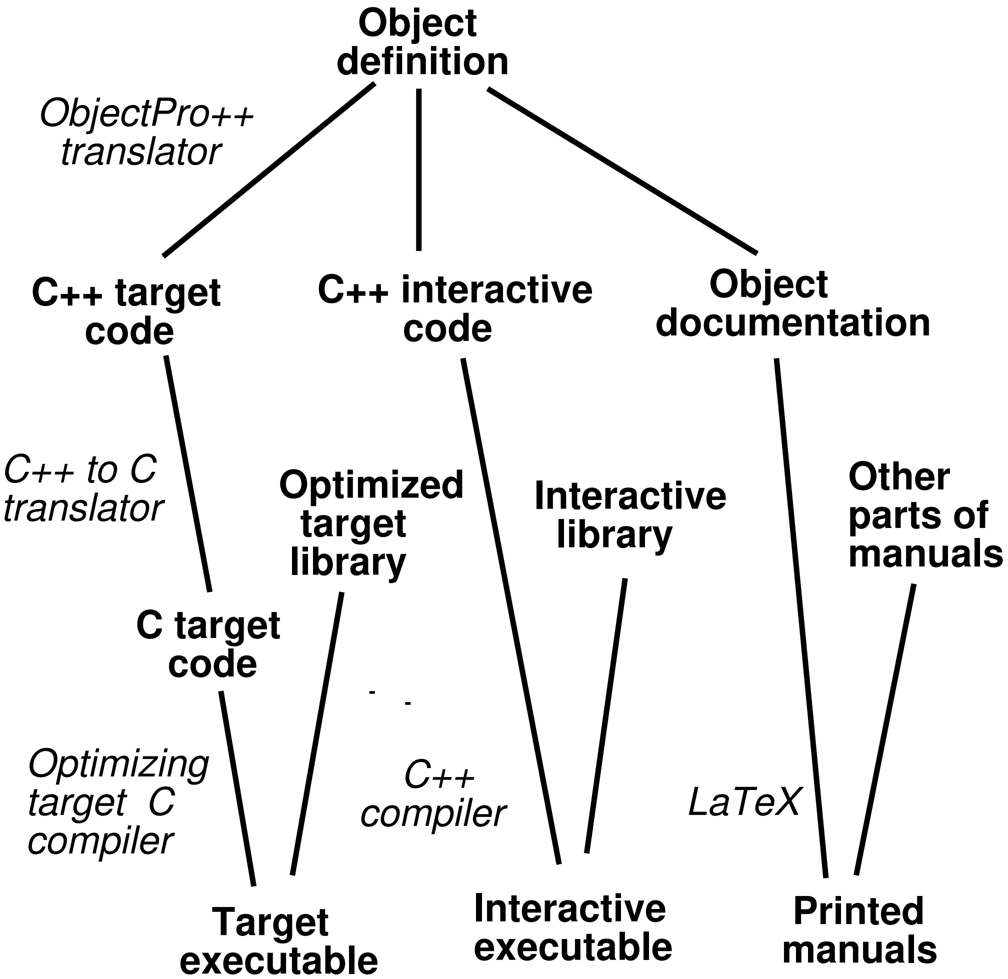


Figure 1: CASE tool for DSP analysis design and implementation

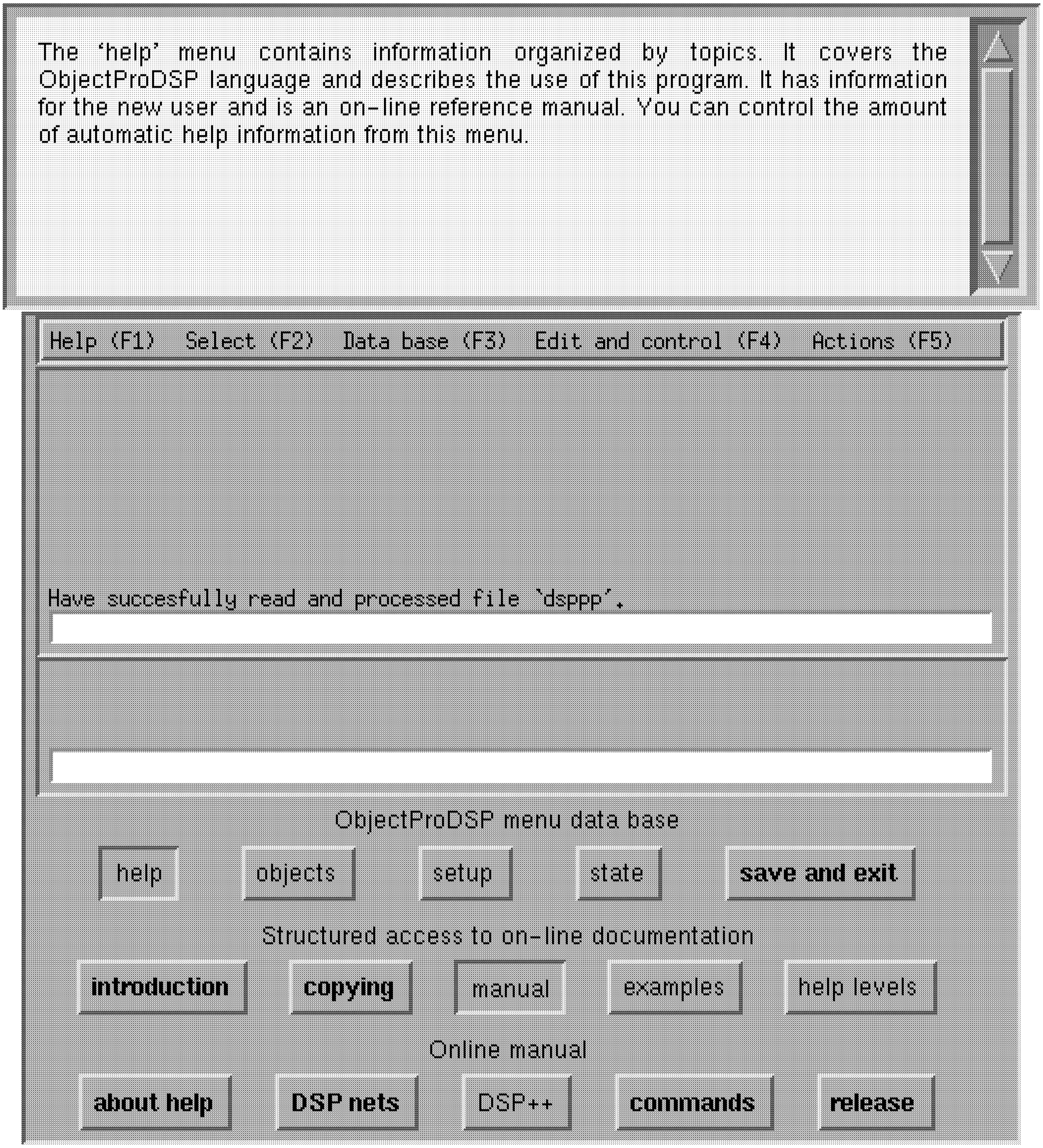
C++ has been extended to define interactive objects and their documentation in the ObjectPro++ language and translator. This allows a single object definition (possibly augmented with optimized assembly language code) to provide the source for interactive execution, printed and interactive user documentation and optimized target code. Figure 1 shows how target and source code and documentation is derived from a single user created source. This facility provides a powerful object oriented DSP CASE tool.

2 Initial windows

Figure 2 shows the initial windows. At the top is a help window that is changed based on your actions. This window can be independently resized. Below this is the main window. At the top is a menubar that documents and controls the use of this window. Below this is space for entering DSP`++` language statements. These are entered in the highlighted area (click in the region to make it the selected point for input). Commands once entered are scrolled into the grey area. This area also contains the language statements that are generated automatically when a network is created graphically.

Below this is a similar area for entering menu data base commands from the keyboard. The last three options selected either with the keyboard or the mouse are scrolled into the grey area. Below this is two or three levels of the menu data base. Push buttons that have text in bold face select a command when button is pushed with the left mouse button. If a push button has normal type it selects another menu. You can 'peel off' any line of the menu tree by holding the `ctrl` key down and clicking mouse button one on a pushbutton. If this is a menu button you will get the first line of this menu in a new window. If this is a command button you will get a copy of the line containing this button in a new window. (These windows can be removed by holding the `ctrl` and `shift` keys down while typing `delete` with the cursor over the window to be deleted.)

All of the shortcut mouse and key options (like holding the control key down and pressing the left mouse button to peel off menus) are listed in the pull down menus at the top of most windows. There is a simple coding scheme for these. Mouse shortcuts (if available) are in square brackets '[ ]'. Keyboard shortcuts are in round brackets '( )'. Keyboard shortcuts are denoted as a single character like '(p)' for lower case 'p' or a word that designates the key like '(space)' for the space bar. A key can be modified with the shift or control keys. '(shift-space)' means hold the shift key down and press the space bar. For the most part you can enter shortcut keys whenever the window is selected and the cursor positioned within it. However there are some 'dead spots' in some windows. If you find that shortcut keys are having no effect move the cursor around.



The 'help' menu contains information organized by topics. It covers the ObjectProDSP language and describes the use of this program. It has information for the new user and is an on-line reference manual. You can control the amount of automatic help information from this menu.

Have succesfully read and processed file `dsppp'.

Figure 2: Initial windows

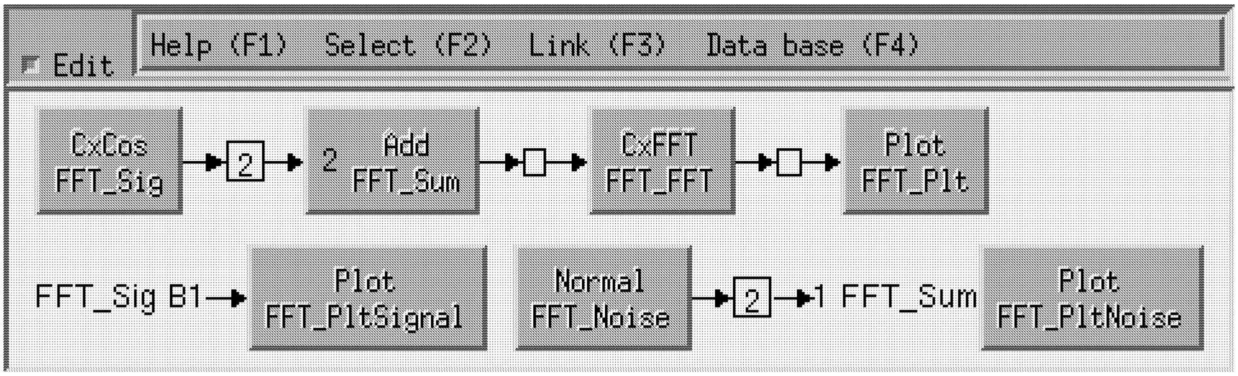


Figure 3: Network from spectral analysis example

3 Spectral analysis example

To select the example described in this section choose `examples` from the second line in the menu. Select `fft` from the third menu line and finally select `execute` from the third menu line. (The main window never has more than three menu lines. Menus you peel off can keep growing up to the length of the screen.)

First a graphical display of the DSP network will appear followed by three windows that contain the network input and output signals. The network display and FFT output plot are shown in figures 3 and 4. All the plots in this example are of complex data. The real part is shown in black and the imaginary part in red (or gray when printed). This is not clear in the printed plots because there is one point per pixel on the default axis. You can see more detail by touching the right arrow key and dragging the mouse between two positions. This will expand the X axis of the entire plot to cover the distance between the X positions you selected. You can see all the options for changing the plot scale by touching the `F3` function key or using the mouse to pull down the `X or Y Detail` menu. Figure 5 shows and expanded plot of the data in figure 4. You can easily see the real and imaginary components. Whenever the plot is expanded to the point where the individual X coordinates are sufficiently separated a dashed line is drawn at each sample plotted. You can see more detail in a plot without changing the scale by making the plot window larger.

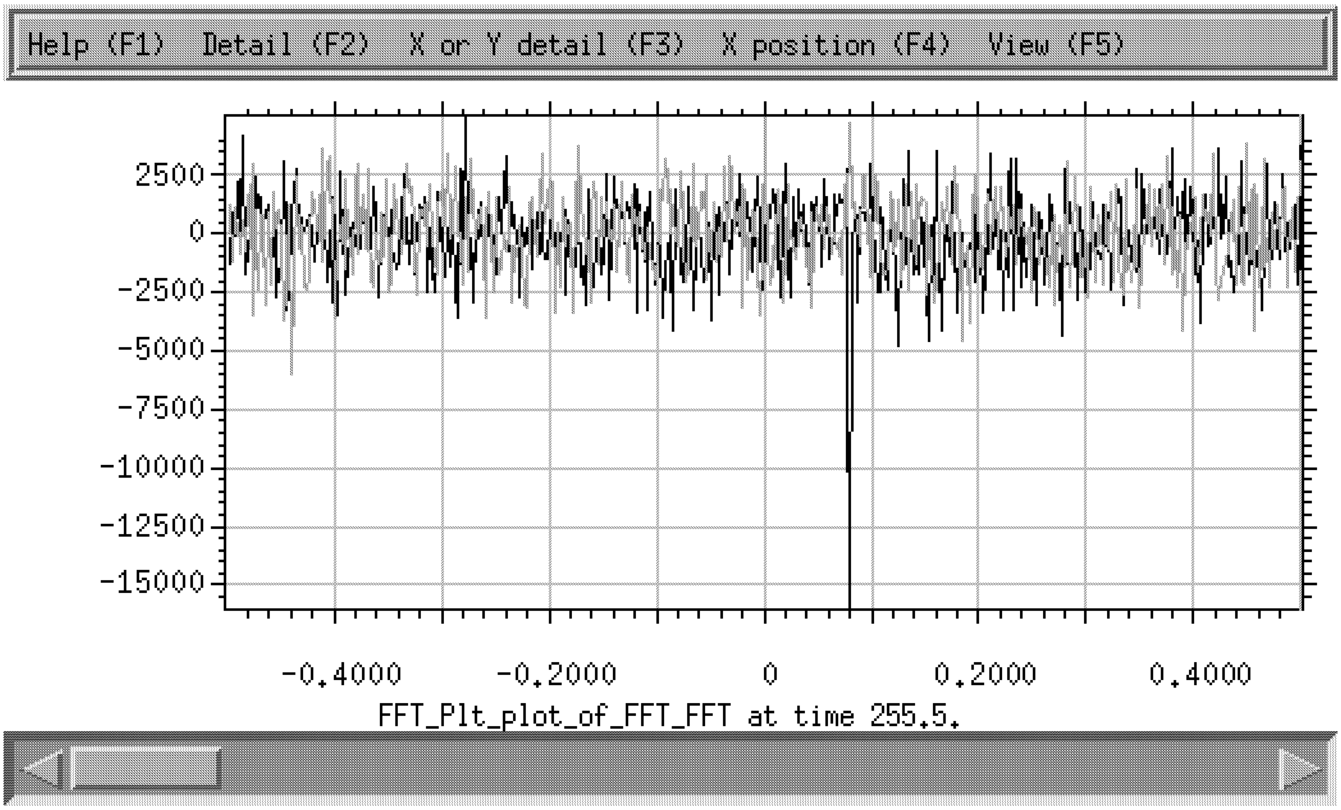


Figure 4: FFT plot from spectral analysis example

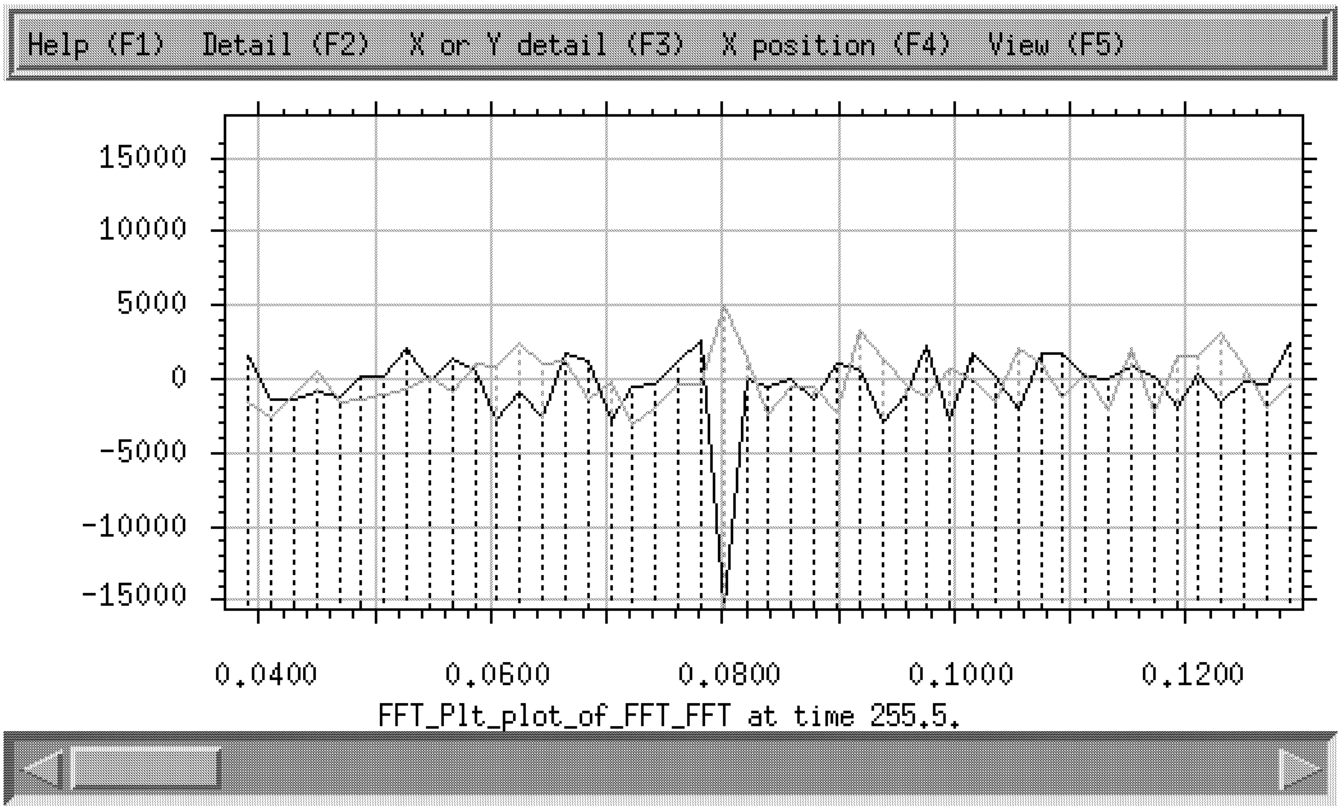


Figure 5: Expanded spectral plot

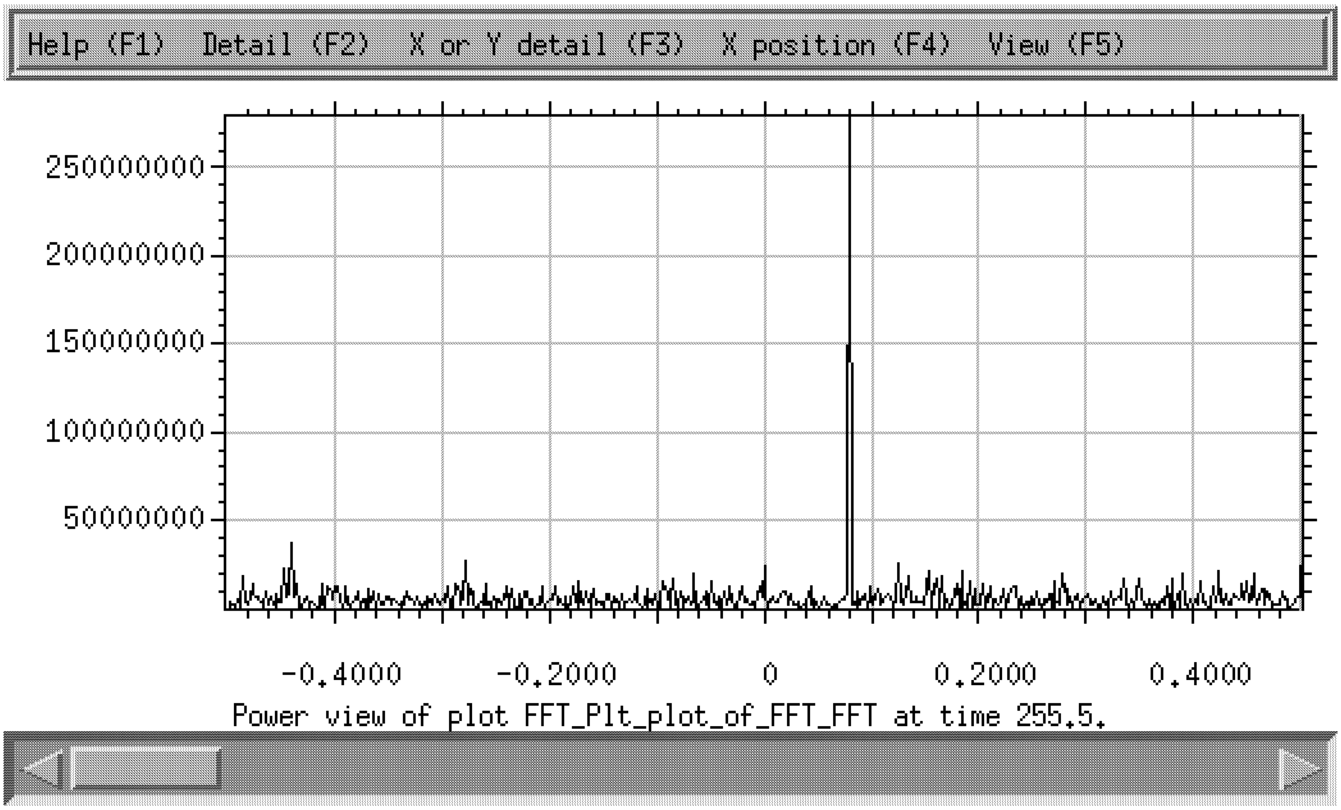
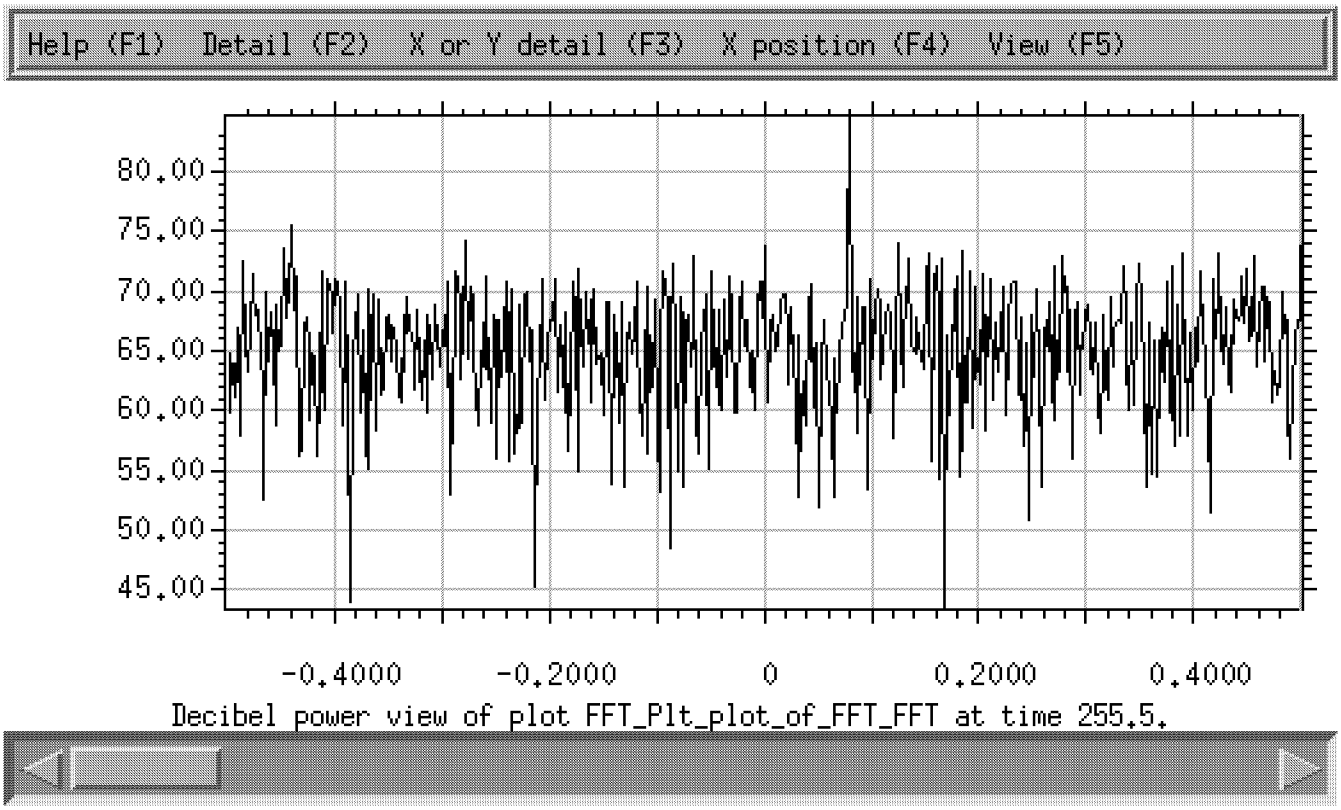


Figure 6: Power spectral plot

For a frequency domain plot such as figure 4, you are often more interested in the power or decibel power. These are both available at one keystroke. Position the cursor in the plot you wish to view in a different way and type `ctrl-p` to see a plot of the power like figure 6 or `ctrl-b` to see a decibel plot like figure 7.

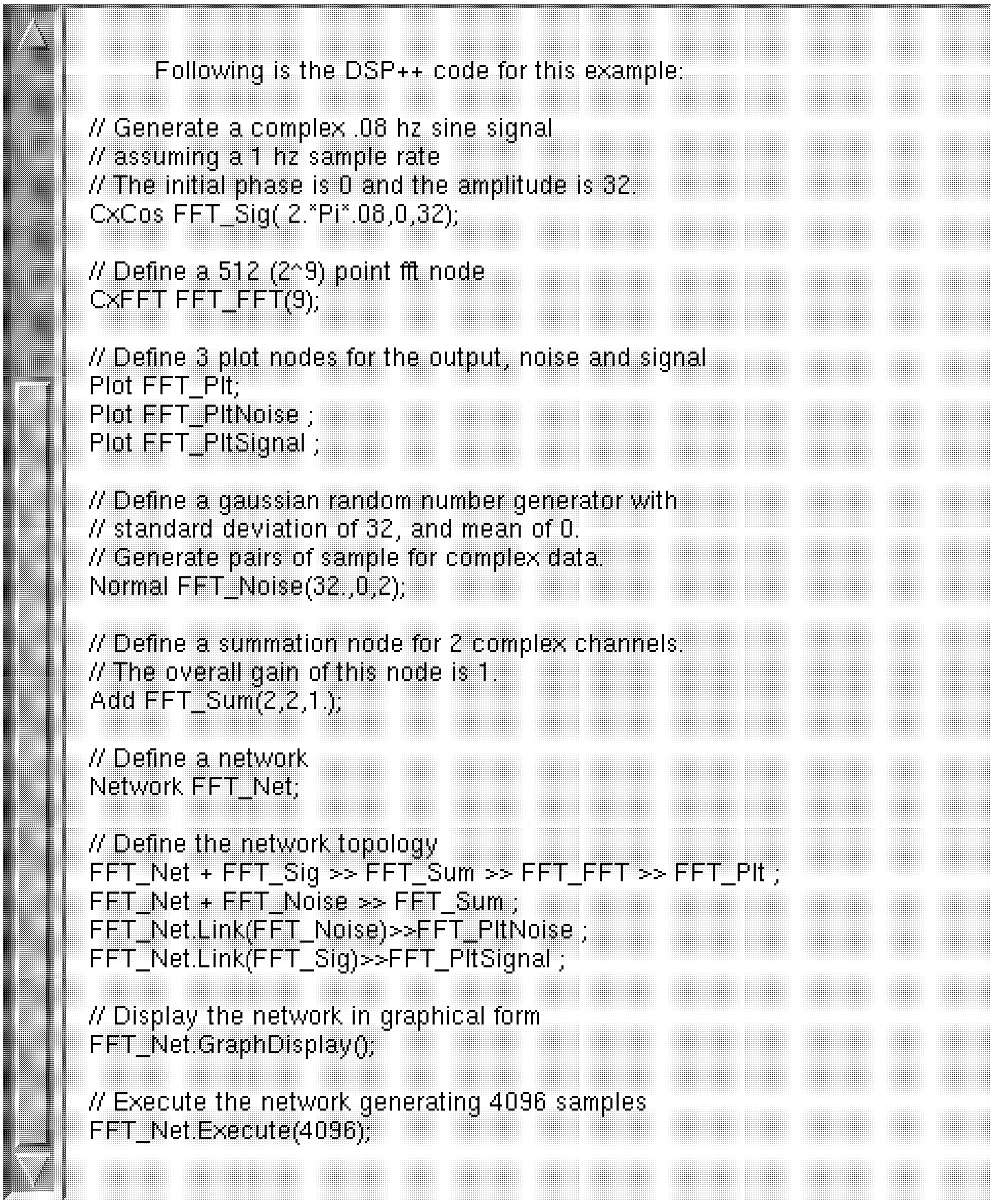
If you now select the `desc` option in the menu for the `fft` example a window will appear with a description and program. Figure 8 shows this enlarged to include the entire program. (To delete help windows like this position the cursor in the window, hold the `ctrl` and `shift` keys down and type `delete`.)

It is not necessary to know the DSP++ language to define DSP networks. They can be built by pointing to the DSP nodes you want to connect. How-



Decibel power view of plot FFT_Plt_plot_of_FFT_FFT at time 255.5.

Figure 7: Decibel spectral plot



Following is the DSP++ code for this example:

```
// Generate a complex .08 hz sine signal
// assuming a 1 hz sample rate
// The initial phase is 0 and the amplitude is 32.
CxCos FFT_Sig( 2.*Pi*.08,0,32);

// Define a 512 (2^9) point fft node
CxFFT FFT_FFT(9);

// Define 3 plot nodes for the output, noise and signal
Plot FFT_Plt;
Plot FFT_PltNoise ;
Plot FFT_PltSignal ;

// Define a gaussian random number generator with
// standard deviation of 32, and mean of 0.
// Generate pairs of sample for complex data.
Normal FFT_Noise(32.,0,2);

// Define a summation node for 2 complex channels.
// The overall gain of this node is 1.
Add FFT_Sum(2,2,1.);

// Define a network
Network FFT_Net;

// Define the network topology
FFT_Net + FFT_Sig >> FFT_Sum >> FFT_FFT >> FFT_Plt ;
FFT_Net + FFT_Noise >> FFT_Sum ;
FFT_Net.Link(FFT_Noise)>>FFT_PltNoise ;
FFT_Net.Link(FFT_Sig)>>FFT_PltSignal ;

// Display the network in graphical form
FFT_Net.GraphDisplay();

// Execute the network generating 4096 samples
FFT_Net.Execute(4096);
```

Figure 8: Program for spectral analysis example

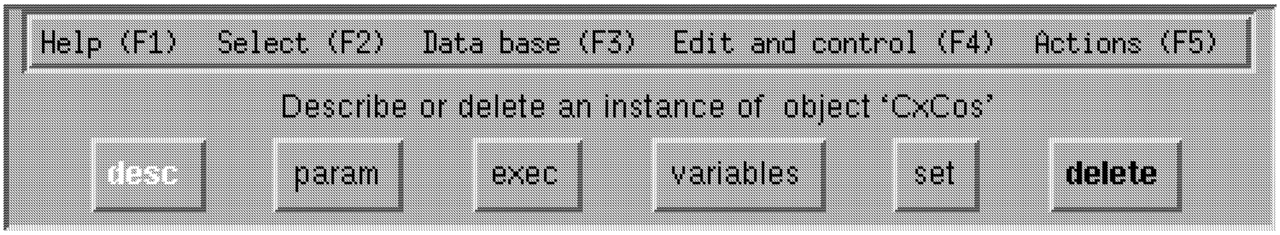
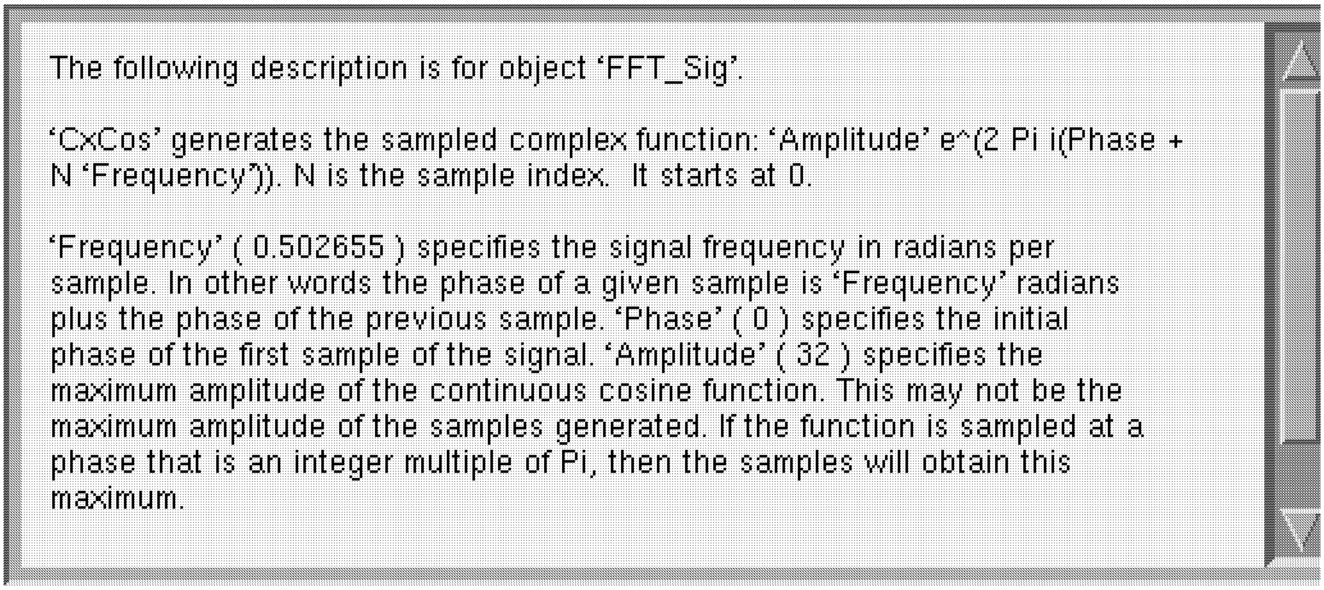


Figure 9: Menu for FFT_Sig object

ever this language provides a concise and readable record of your work. It is the format used for saving objects and networks between sessions. It may be helpful to refer to both the program in figure 8 and the network display in figure 3 in understanding this example.

The nodes in the network display have two names. The top one is the class name and the lower one is the object name. The first node is a `CxCos` or complex cosine signal called `FFT_Sig`. We can refer to the program listing to see the frequency, phase and amplitude of this signal. We can also get this information by holding the `ctrl` key down and clicking the left mouse button with the cursor in this node. The menu that comes up when you do this and the description that will appear in the help window when you select `desc` from this menu are in figures 9 and 10. We see from this figure that the frequency is 0.02655 radians per sample. From the program we see that this corresponds to a frequency of $2 \times \pi \times .08$ radians per sample or .08 hz with a 1 hz sample rate. A default sample rate of 1 hz. is assumed. You can set the sample rate for any node in the network and the timing will be adjusted for all affected displays. Select the `exec` option from the menu for `FFT_Sig` and then the `SetSampleRate` option to change the sample rate of the network. If you check the spectral plot in figure 4 you will see there is a peak at .08 hz. If you now change the sample rate to 4 hz the scale on the frequency plot will be adjusted so the peak is at .32 hz.

Returning to the display of the entire network in figure 3 we see that `FFT_Sig` is connected to a small box with the number '2' in it. This represents an output buffer that drives the inputs of two nodes. One of these is node `FFT_Sum`. The '2' in the left hand part of this node stands for two input channels. Each node can have a variable number of input channels, output



The following description is for object ‘FFT_Sig’.

‘CxCos’ generates the sampled complex function: ‘Amplitude’ e^(2 Pi i(Phase + N ‘Frequency’)). N is the sample index. It starts at 0.

‘Frequency’ (0.502655) specifies the signal frequency in radians per sample. In other words the phase of a given sample is ‘Frequency’ radians plus the phase of the previous sample. ‘Phase’ (0) specifies the initial phase of the first sample of the signal. ‘Amplitude’ (32) specifies the maximum amplitude of the continuous cosine function. This may not be the maximum amplitude of the samples generated. If the function is sampled at a phase that is an integer multiple of Pi, then the samples will obtain this maximum.

Figure 10: Description of FFT_Sig object

channels and nodes driven by each output channel. These are represented by numbers at the left of the node at the right of the node and in the buffer box to the right of the node. The output from `FFT_Sum` goes to a complex FFT (`CxFFT`) called `FFT_FFT` and a plotting node called `FFT_Plot`. This latter node generated the spectral plot we just discussed. The other output from `FFT_Sig` goes to another plotting node called `FFT_PltSignal`. Notice that the plots for both the spectral output and the linear time series are the same class `Plot`. This class of node knows the type of data it is receiving and generates the correctly formatted plot. You need not select different nodes or enter parameters redundantly to get the correct plot. The other input to `FFT_Sum` comes from a Gaussian noise generator `FFT_Noise`. The output from this node is also plotted.

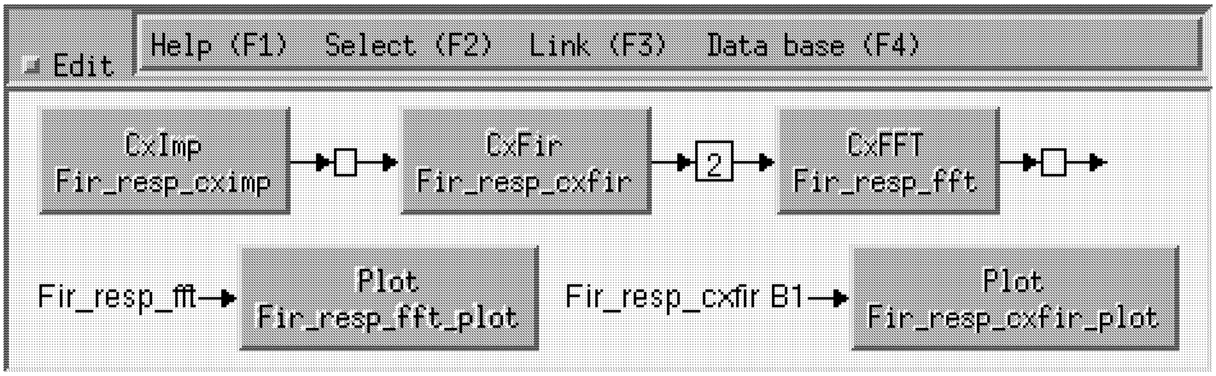


Figure 11: Network to compute FIR filter response

4 FIR filter frequency response

Figure 11 shows a network to compute and display the frequency response of a Finite Impulse Response (FIR) filter. A complex impulse signal (with real part 0) is passed through a FIR filter and a complex FFT. The outputs of the FIR filter and FFT are plotted. Figures 12 through 14 show complex magnitude, power and decibel views of the frequency response. The latter two plots were obtained by typing `ctrl-p` and `ctrl-b` with the cursor over the first plot. The program for this example is in in figure 17.

The steep descent at the right edge of Figure 14 is a consequence of processing the even symmetric impulse repsonse of the FIR filter with the FFT. The largest/smallest frequency bin convolves the signal with $e^{\pi i n}$ where n is the sample index. This is the real sequence $\{-1, 1, -1, 1, ...\}$. Convolving this sequence with any even symmetric signal produces 0. The decibel plot artificially limits the smallest value to -200 db.

The impulse signal generator used in this example supports special signals. You can hold the `ctrl` key down and click the left mouse button on the `Fir_resp_cximp` object in the network display to get the menu for this object. Select `desc` from this menu to get a description of the parameters and their values in this example. This menu is in figure 15 and the description is in figure 16.

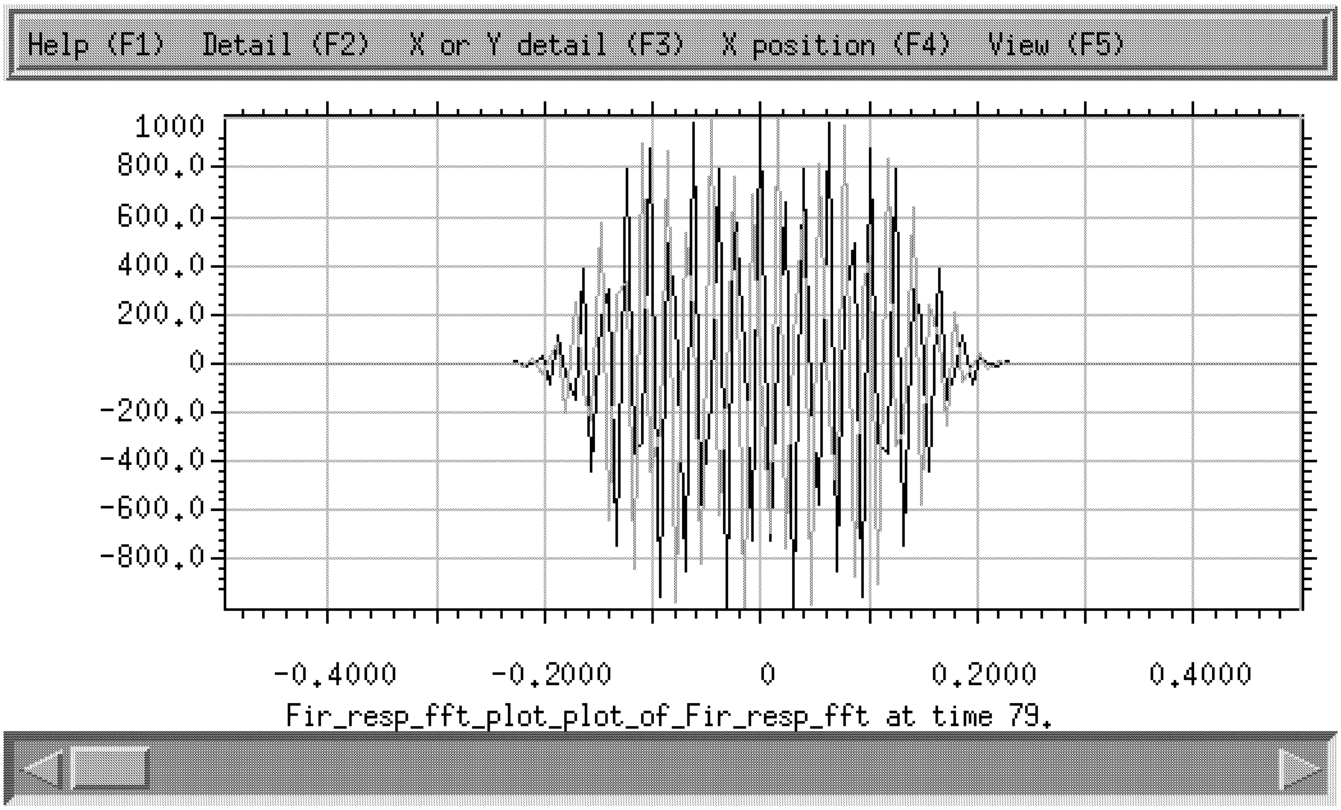
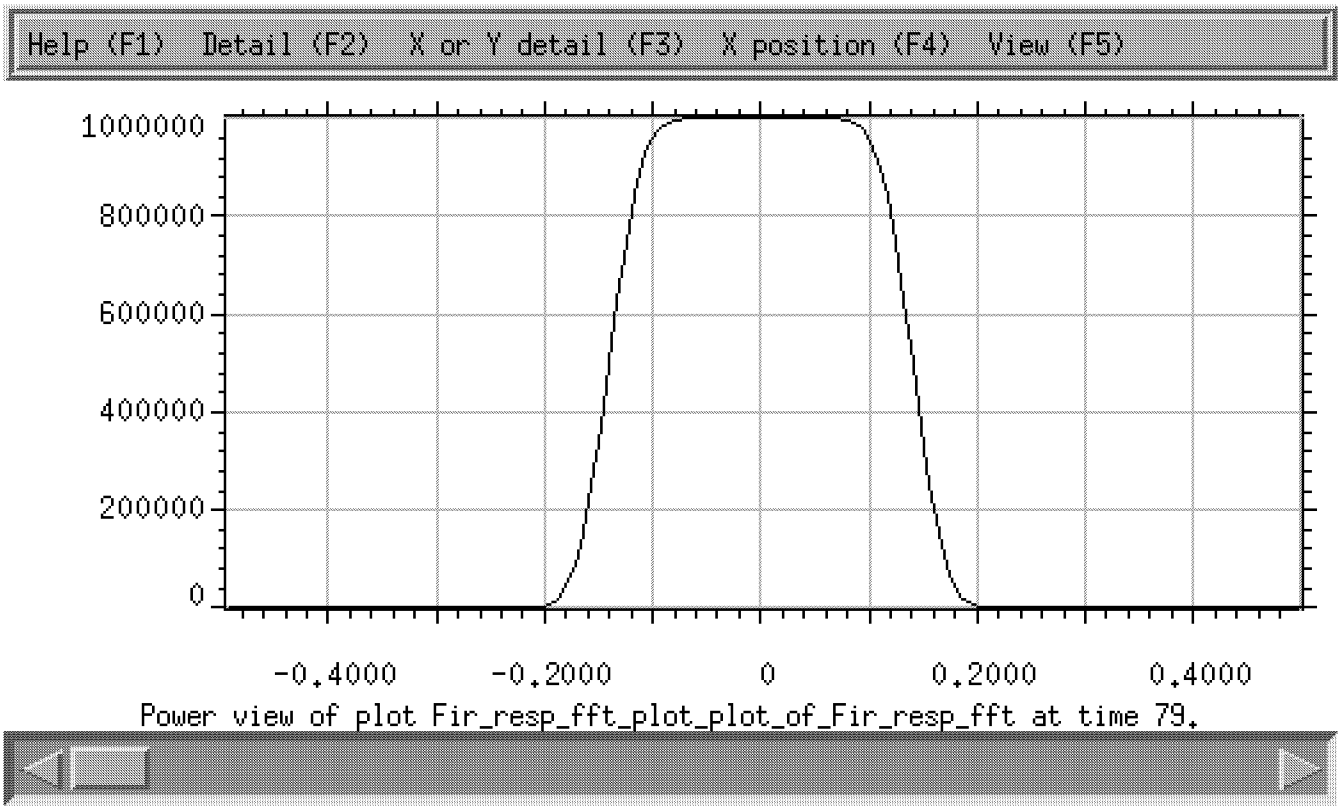
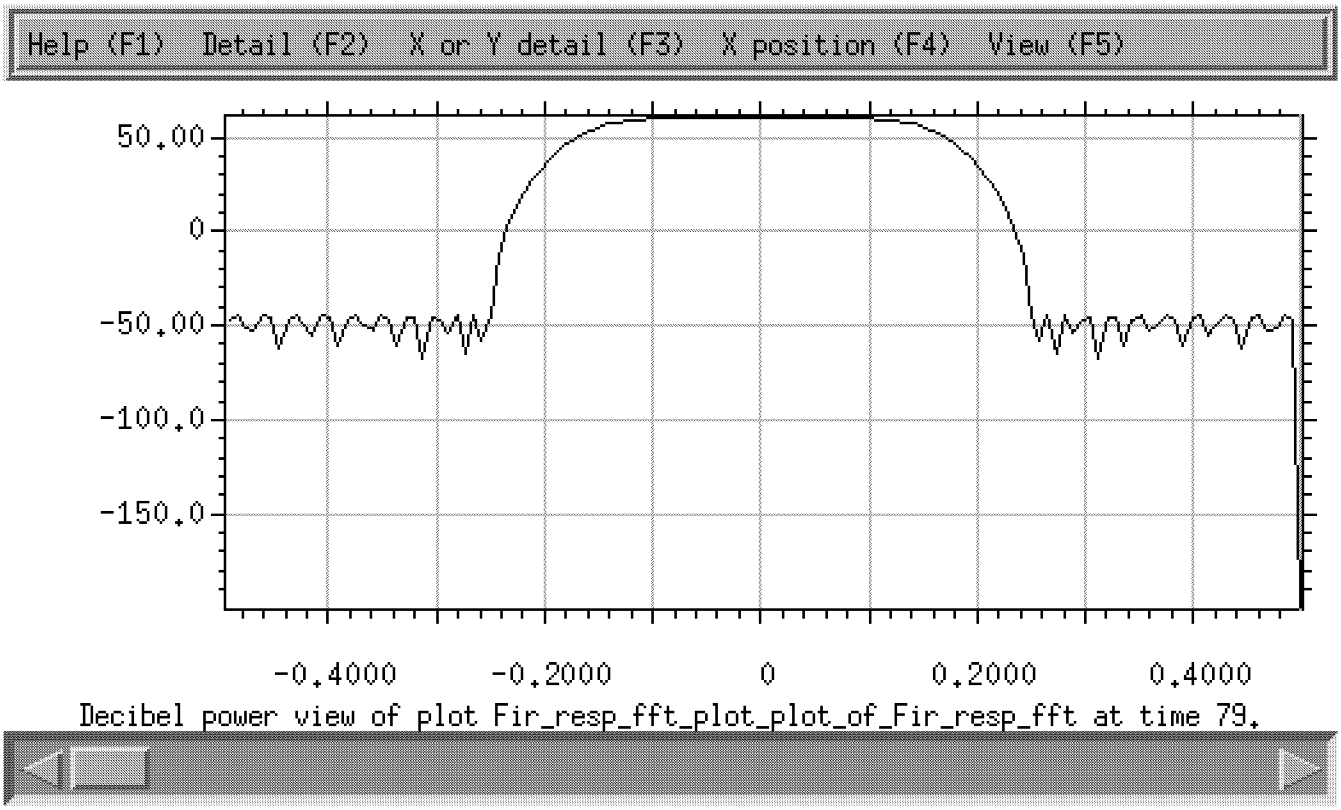


Figure 12: Complex plot of FIR filter frequency response



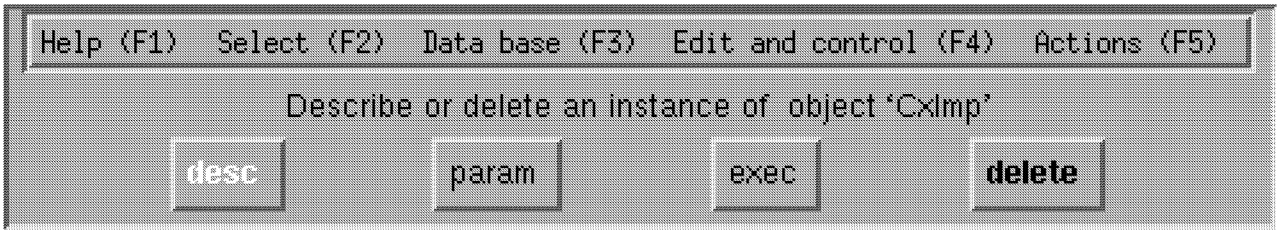
Power view of plot Fir_resp_fft_plot_plot_of_Fir_resp_fft at time 79.

Figure 13: Power plot of FIR filter frequency response



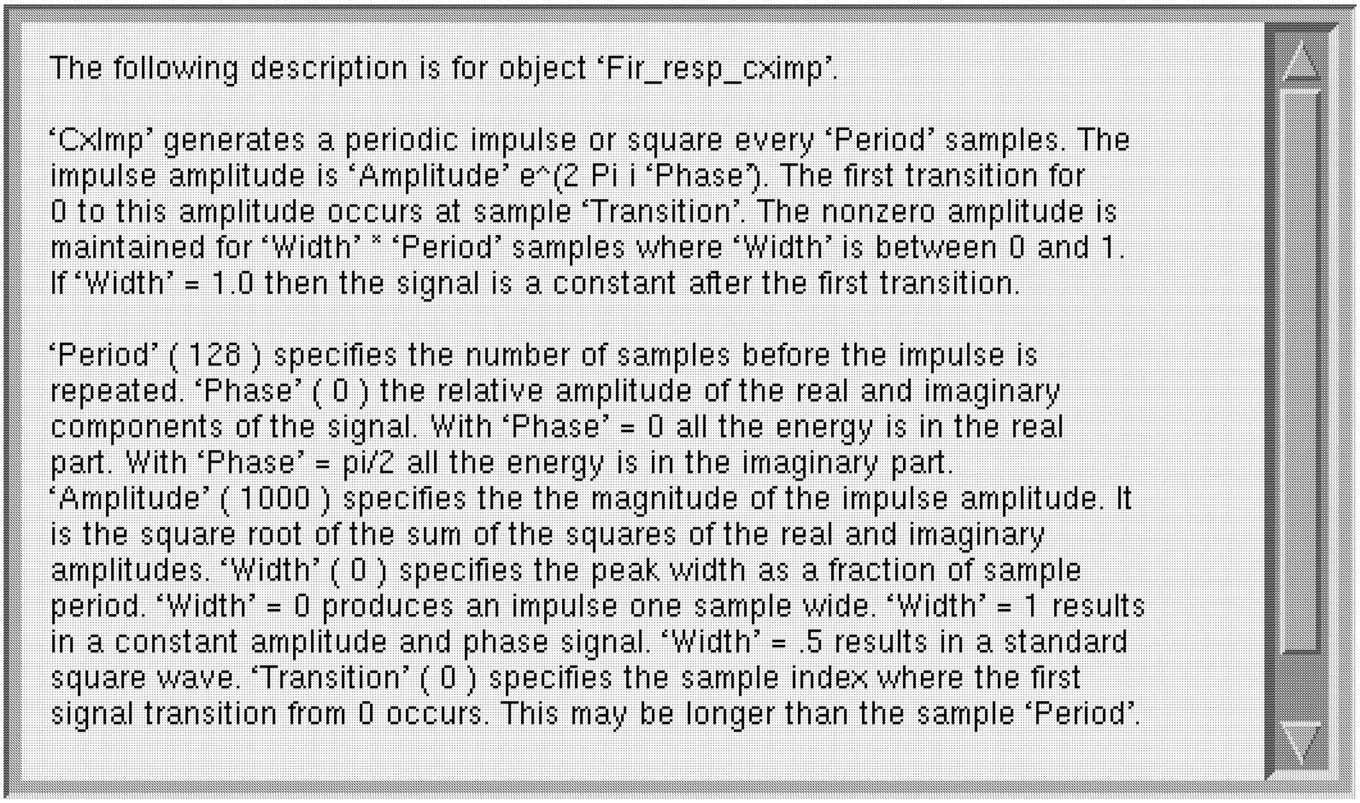
Decibel power view of plot Fir_resp_fft_plot_plot_of_Fir_resp_fft at time 79.

Figure 14: Decibel plot of FIR filter frequency response



Describe or delete an instance of object 'CxImp'

Figure 15: Menu for Fir_resp_cximp object

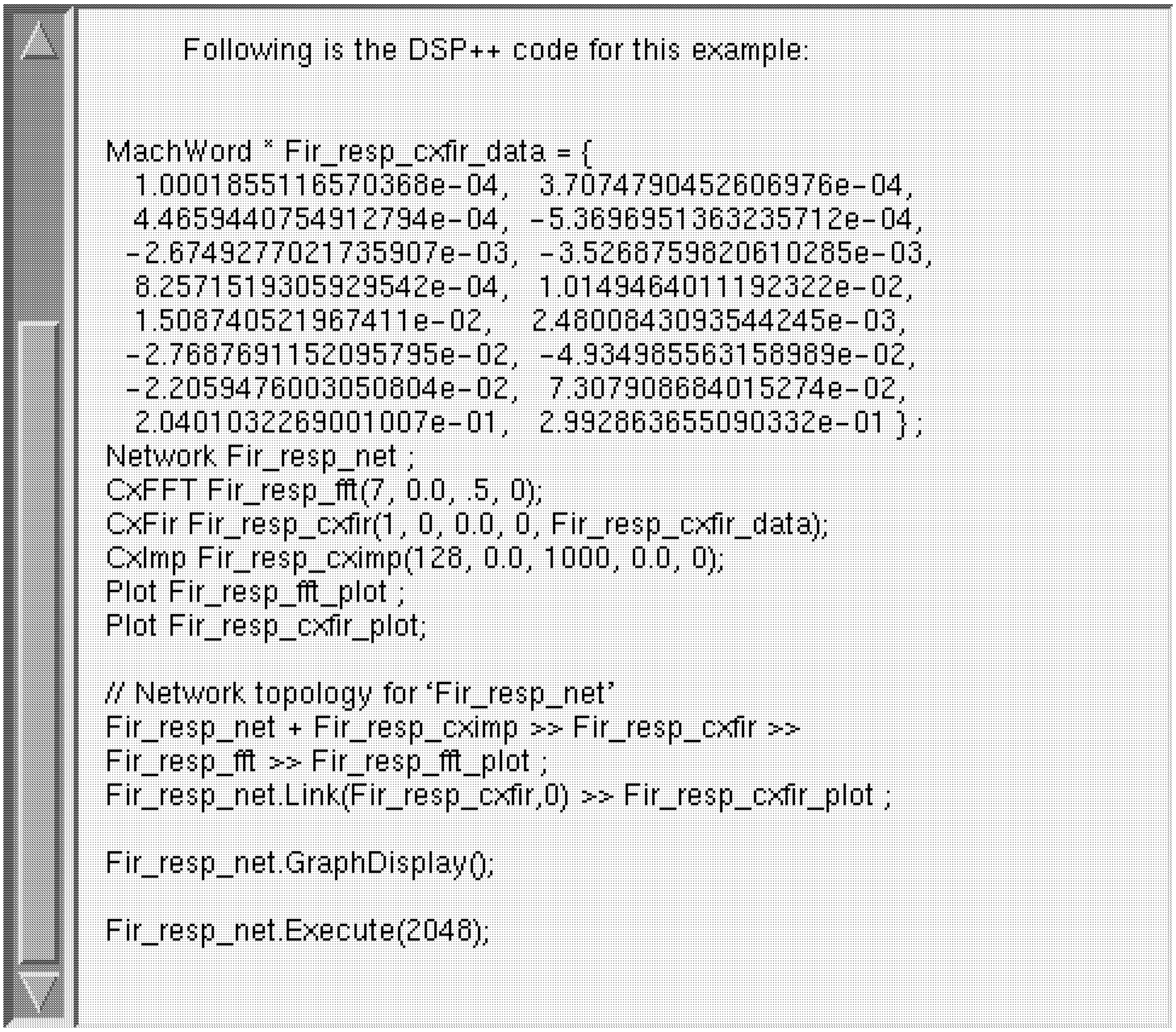


The following description is for object ‘Fir_resp_cximp’.

‘CxImp’ generates a periodic impulse or square every ‘Period’ samples. The impulse amplitude is ‘Amplitude’ e^(2 Pi i ‘Phase’). The first transition for 0 to this amplitude occurs at sample ‘Transition’. The nonzero amplitude is maintained for ‘Width’ * ‘Period’ samples where ‘Width’ is between 0 and 1. If ‘Width’ = 1.0 then the signal is a constant after the first transition.

‘Period’ (128) specifies the number of samples before the impulse is repeated. ‘Phase’ (0) the relative amplitude of the real and imaginary components of the signal. With ‘Phase’ = 0 all the energy is in the real part. With ‘Phase’ = pi/2 all the energy is in the imaginary part. ‘Amplitude’ (1000) specifies the the magnitude of the impulse amplitude. It is the square root of the sum of the squares of the real and imaginary amplitudes. ‘Width’ (0) specifies the peak width as a fraction of sample period. ‘Width’ = 0 produces an impulse one sample wide. ‘Width’ = 1 results in a constant amplitude and phase signal. ‘Width’ = .5 results in a standard square wave. ‘Transition’ (0) specifies the sample index where the first signal transition from 0 occurs. This may be longer than the sample ‘Period’.

Figure 16: Description of Fir_resp_cximp object



Following is the DSP++ code for this example:

```
MachWord * Fir_resp_cxfir_data = {
  1.000185511657036 8e-04,  3.7074790452606976e-04,
  4.4659440754912794e-04, -5.3696951363235712e-04,
 -2.6749277021735907e-03, -3.5268759820610285e-03,
  8.2571519305929542e-04,  1.0149464011192322e-02,
  1.508740521967411e-02,   2.4800843093544245e-03,
 -2.7687691152095795e-02, -4.9349855631589 89e-02,
 -2.2059476003050804e-02,  7.307908684015274e-02,
  2.0401032269001007e-01,  2.992863655090332e-01 } ;
Network Fir_resp_net ;
CxFFT Fir_resp_fft(7, 0.0, .5, 0);
CxFir Fir_resp_cxfir(1, 0, 0.0, 0, Fir_resp_cxfir_data);
CxImp Fir_resp_cximp(128, 0.0, 1000, 0.0, 0);
Plot Fir_resp_fft_plot ;
Plot Fir_resp_cxfir_plot;

// Network topology for 'Fir_resp_net'
Fir_resp_net + Fir_resp_cximp >> Fir_resp_cxfir >>
Fir_resp_fft >> Fir_resp_fft_plot ;
Fir_resp_net.Link(Fir_resp_cxfir,0) >> Fir_resp_cxfir_plot ;

Fir_resp_net.GraphDisplay();

Fir_resp_net.Execute(2048);
```

Figure 17: Program for FIR filter response

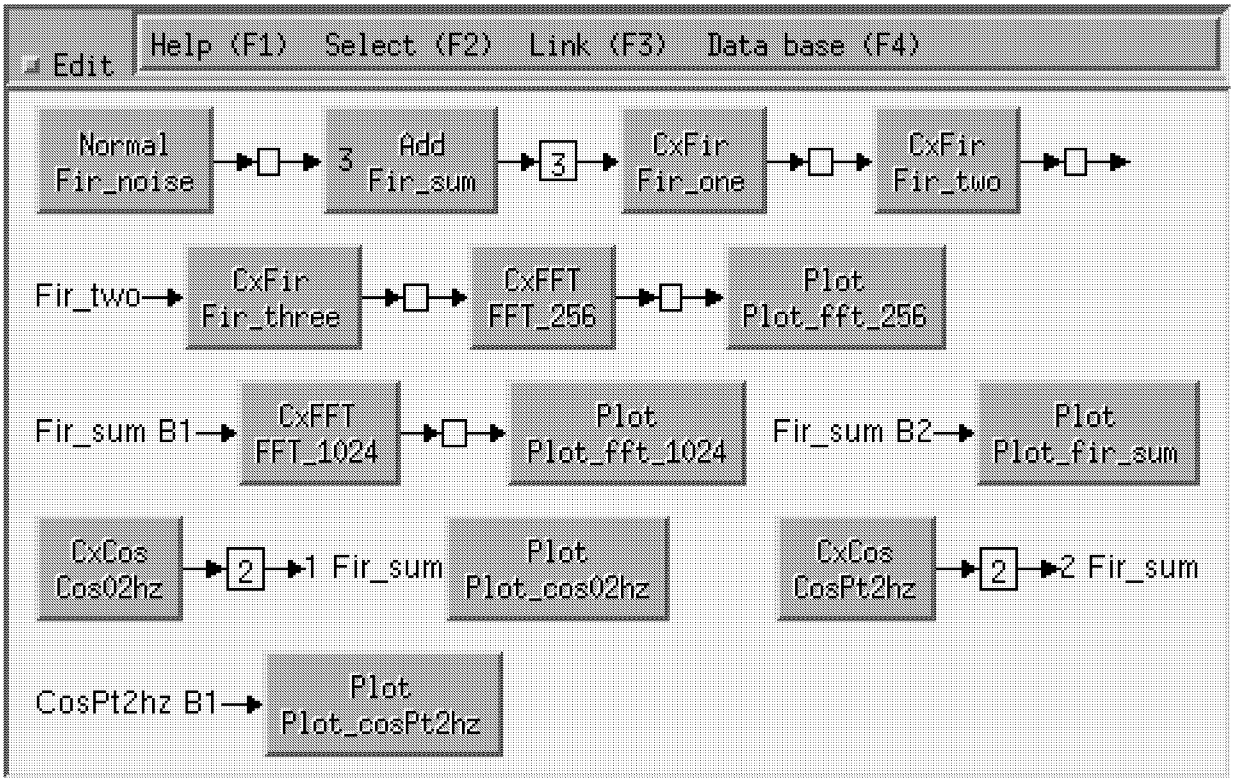
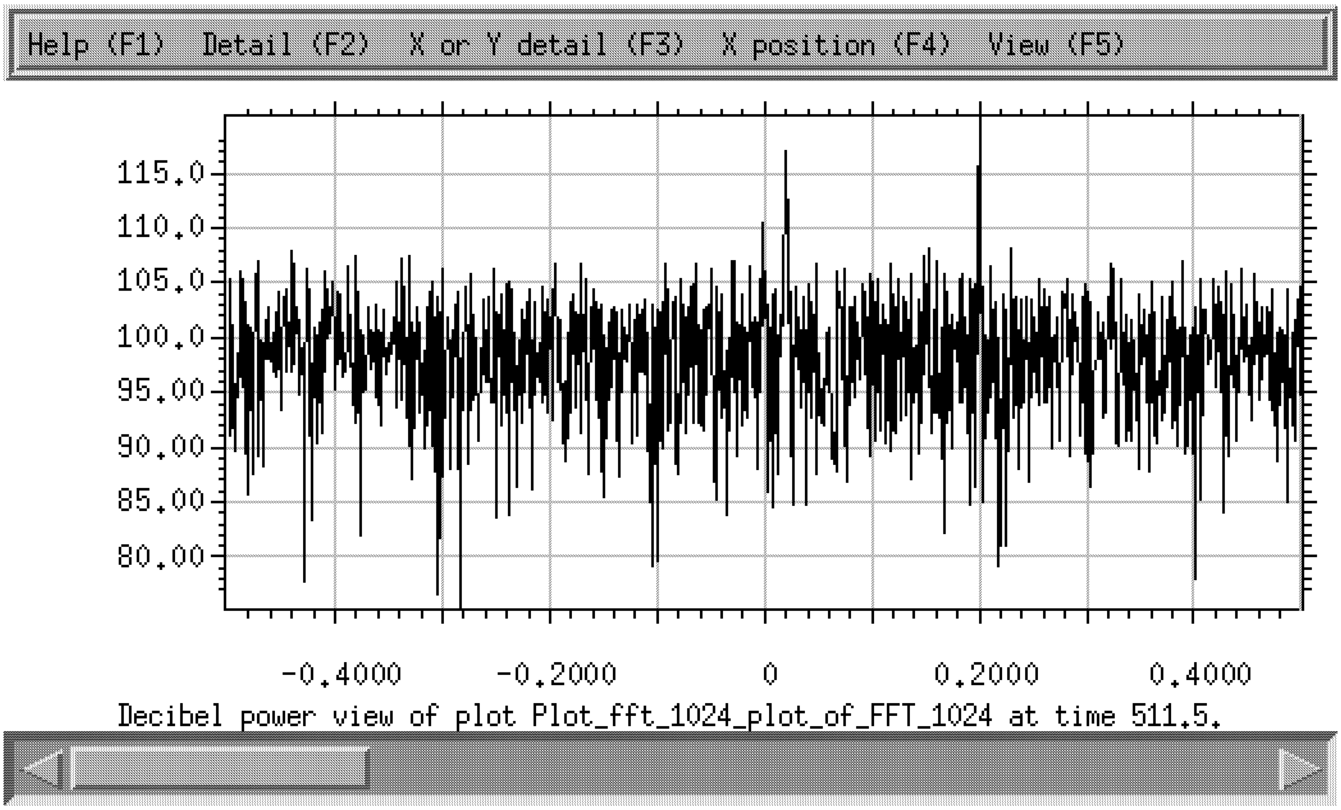


Figure 18: Network for multiple stage filter

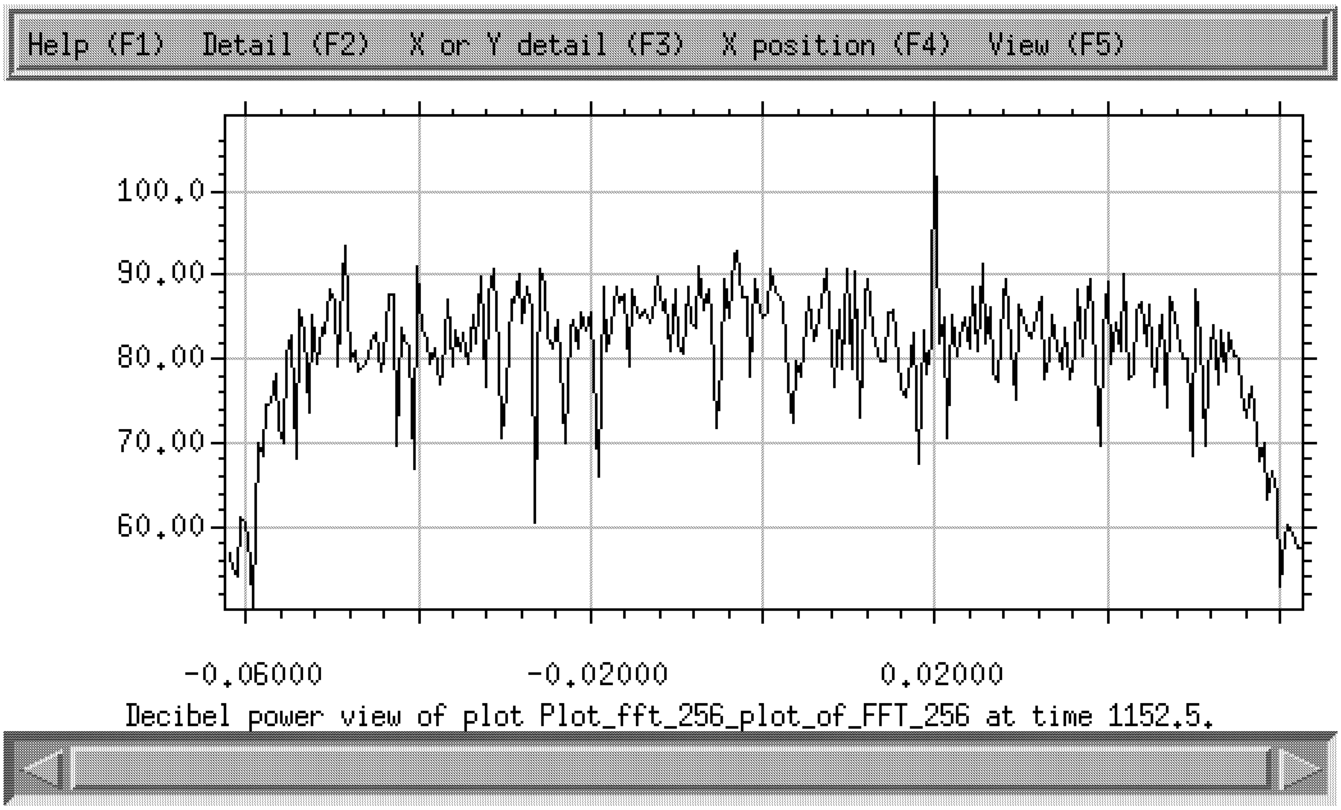
5 Multi-stage FIR filter

Figure 18 is a spectral analysis example using a three stage filter. Each stage has a resampling factor of 2. This structure can significantly reduce the total number of multiplies over that required for a single stage filter with the same resampling ratio. The input is Gaussian noise and two complex sine waves at frequencies of .2 and .02 hz. Both of the cosine signals are visible in the spectral plot of the filter input in figure 19. The higher frequency signal is outside the pass band of the filter as seen in the filter output spectral plot in figure 20. The frequency axis of these plots are on the same scale. ObjectProDSP automatically computes the sampling rate ratios of every node in a network and uses these ratios in labeling the plots.



Decibel power view of plot Plot_fft_1024_plot_of_FFT_1024 at time 511.5.

Figure 19: Spectral plot of multi-stage filter input



Decibel power view of plot Plot_fft_256_plot_of_FFT_256 at time 1152.5.

Figure 20: Spectral plot of multi-stage filter output
]

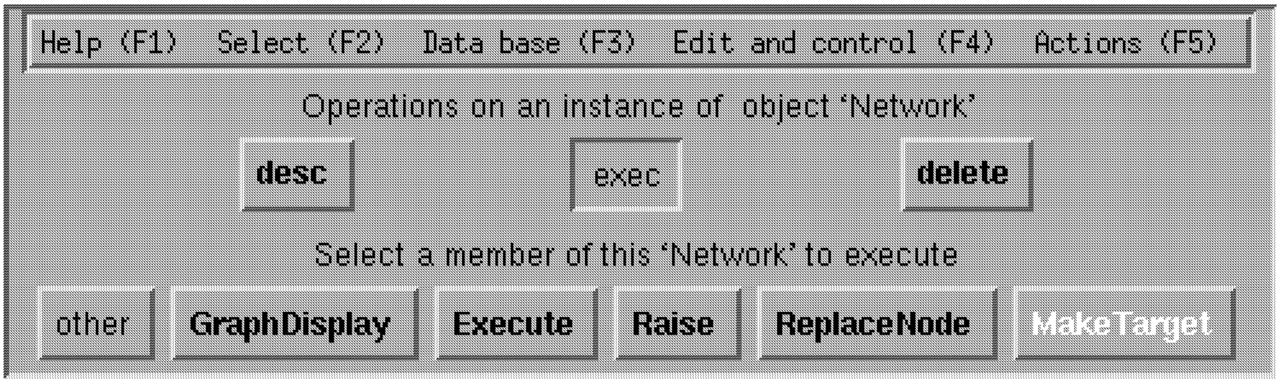


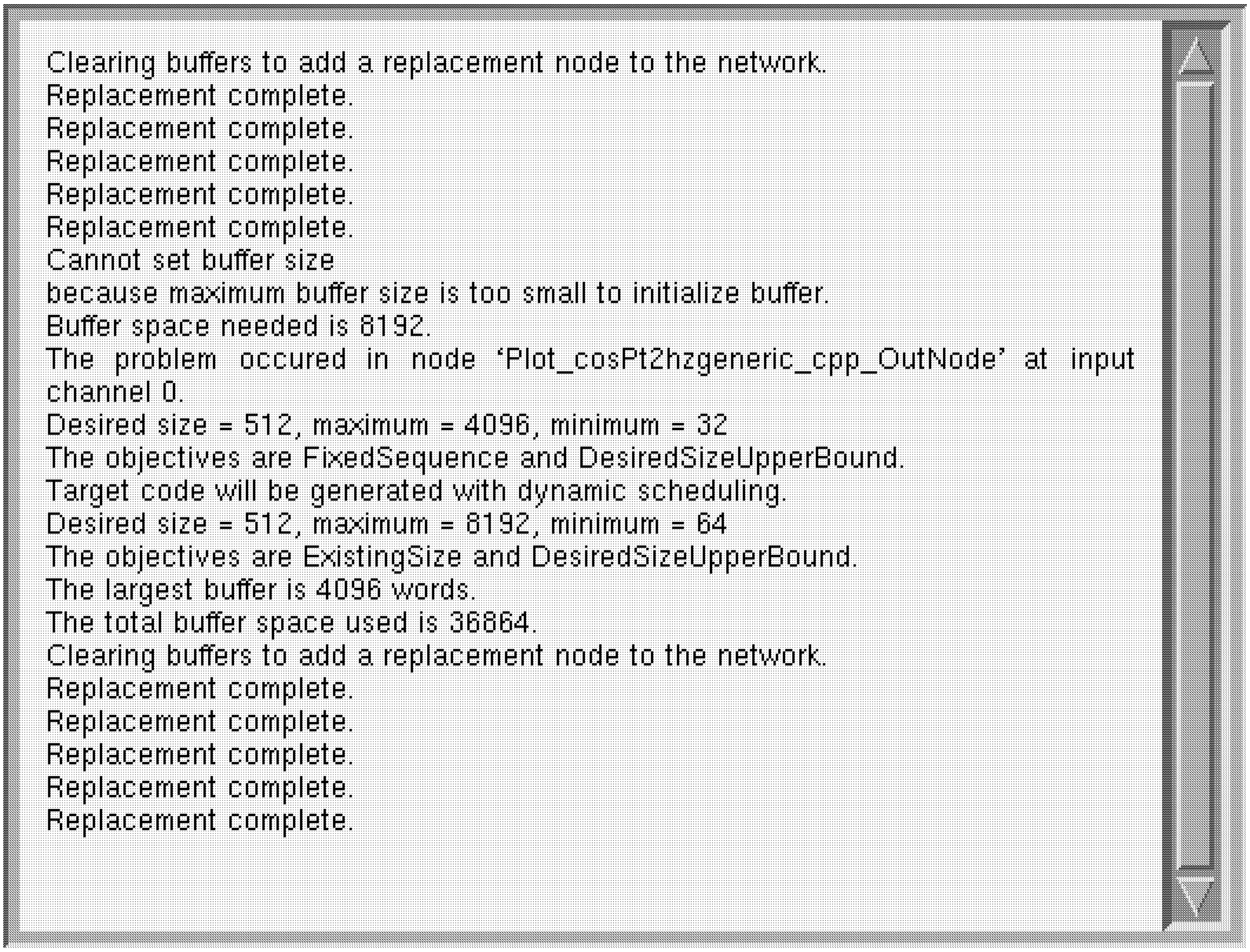
Figure 21: Menu for Fir_net object

To create code for a supported target position the cursor over the bar containing the edit button in the `FFT_net` window and click the left mouse button while holding the `ctrl` key down. Select the `exec` and then the `other` options in the `Menu Fir_net` window that appears. Now select `MakeTarget`. This menu is in figure 21.

You will now be prompted for three parameters. First is the target type. If you select the default `generic_cpp` code will be generated to run on the computer you are using. This version only provides direct support for generic C++ code. Next enter '1' to compile and link the code to be generated. If you select 0 you can compile and link it later by doing a `make` in a subdirectory of the one in which the code will be written. Finally you are prompted for a directory name. If you select the default, a directory will be created under your current working directory that has the same name as the network you are making the target for.

Code will be written and a make file generator process will be started. The output from this and execution of the make (if you selected create) will be written to file `msgs_FFT_Net_FFT_Net.out` in your current directory. The `Makefile` and the executable image will be in subdirectory DppNmtarflt of the directory you specified.

The result of analyzing the network and generating the target code is in the 'help information' window shown in figure 22. The `Replacement complete` lines refer to the replacement of nodes not supported on the selected target.



```
Clearing buffers to add a replacement node to the network.
Replacement complete.
Replacement complete.
Replacement complete.
Replacement complete.
Replacement complete.
Cannot set buffer size
because maximum buffer size is too small to initialize buffer.
Buffer space needed is 8192.
The problem occured in node 'Plot_cosPt2hzgeneric_cpp_OutNode' at input
channel 0.
Desired size = 512, maximum = 4096, minimum = 32
The objectives are FixedSequence and DesiredSizeUpperBound.
Target code will be generated with dynamic scheduling.
Desired size = 512, maximum = 8192, minimum = 64
The objectives are ExistingSize and DesiredSizeUpperBound.
The largest buffer is 4096 words.
The total buffer space used is 36864.
Clearing buffers to add a replacement node to the network.
Replacement complete.
Replacement complete.
Replacement complete.
Replacement complete.
Replacement complete.
```

Figure 22: Target code generation with large buffers

For example no targets support nodes for plotting. These are replaced by nodes that write a file that can be read and plotted later.

First an attempt is made to create a deterministic scheduling scheme for the network. This is a a list of what nodes to execute in what order and how many samples to process at each execution. If this requires excessive buffer space or the lists become too long a dynamic scheduling scheme is used.

If you want to change the buffer sizes or other parameters of this scheduling process you can assign a new buffer descriptor to the network. First click the mouse on the `Edit` button of the network you want to change. Then enter the command `network` in the menu database text entry window or make the menu selections `objects` and `network`. Hold the `shift` key down and click the right mouse button over the `CircBufDes` button. You will be prompted for the new buffer parameters. You can now select `MakeTarget` to generate code based on these parameters.

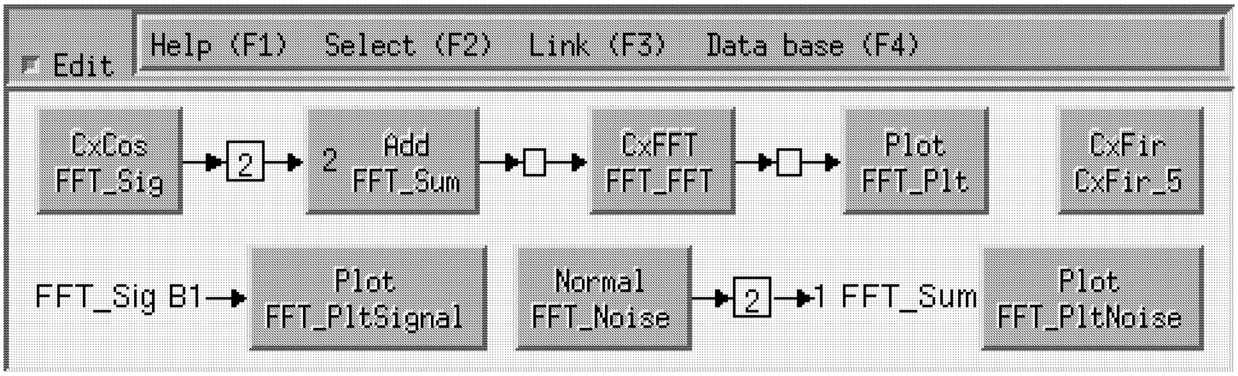


Figure 23: Editing a CxFir node into FFT_Net.

6 Constructing and editing DSP networks

You can add additional nodes to any of the example networks. Assume you want to add a thread to the `FFT_Net` example that filters the output of the `FFT_Sum` node with the filter response in the `Fir_resp` example. First execute the `fft` example to get a display of `FFT_Net`. Click the left mouse button with the cursor over the `Edit` push button at the upper left corner of this display. A check mark should appear. Now select `objects` and `dsp processing` from the first and second lines in the menu database. The FIR filter we want uses the default coefficients. To get an instance of the filter with the default parameters click the left mouse button with the `shift` key depressed and the cursor over `CxFir` in the third line of the menu data base. A `CxFir` node should appear in the `Fir_net` window as in figure 23. This node is not connected yet.

To complete the network and get useful output we need an FFT and a plotting node. We can add all these to the network display before connecting them. The default FFT size is only 16 points so we want to change this parameter from its default value. To add an FFT node and set its parameters hold the `shift` key down while clicking the *right* mouse button with the cursor over the `CxFFT` button in the third row of the menu data base. You will be prompted for a series of parameters. We only want to change the size of the FFT or the second parameter. You can type return to select the default value of the others. For the second parameter you must specify the

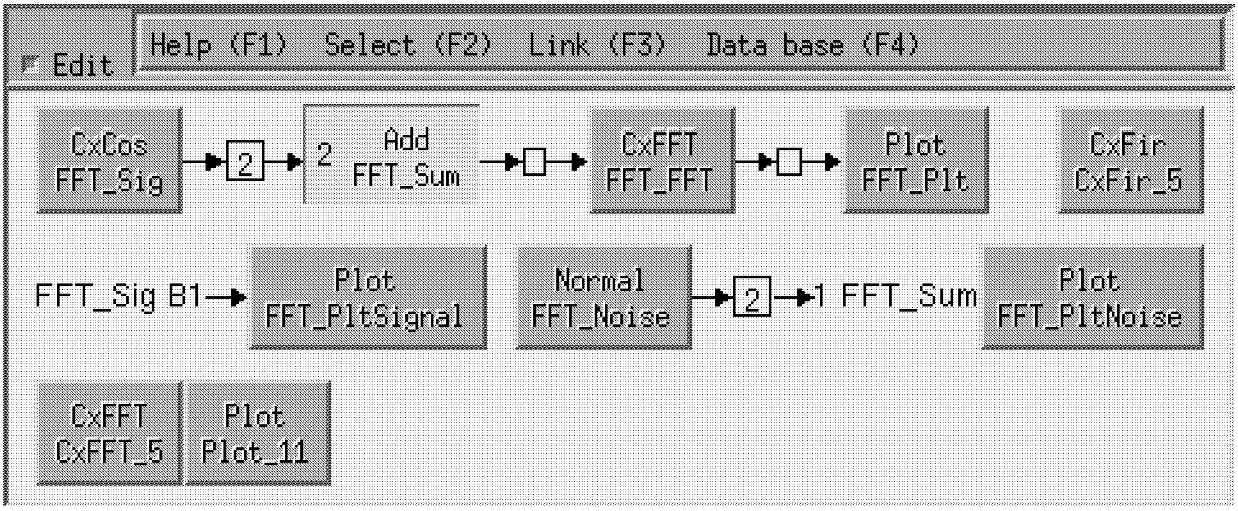


Figure 24: Selecting FFT_Sum to connect to.

log base 2 of the FFT size. To make this comparable to the FFT already in this example select 9 for an FFT size of 512. Now you should have two disconnected nodes in the `Fir_net` display. To add a plotting node select `plot` from the second line of the menu data base and and then hold the `shift` key down and click the left mouse button with the cursor over the `Plot` button in the third line of the menu data base.

To link these nodes in the network you need to understand how to reference the three points where a node can be connected. Input channels are referenced with the left mouse button, output channels are referenced with the middle mouse button and output buffers are reference with the right mouse button. The distinction between output buffers and output channels is important. Nodes have a fixed number of output channels, they can however have any number of connection from the output buffer associated with *each* output channel. The outputs of the different output channels of a node are usually different data streams. For example the `Demux` node demultiplexes a single data stream into multiple data streams that each contain different samples from the input. The multiple outputs from a single buffer all get the same data.

In editing a network, selecting an output channel selects the next available

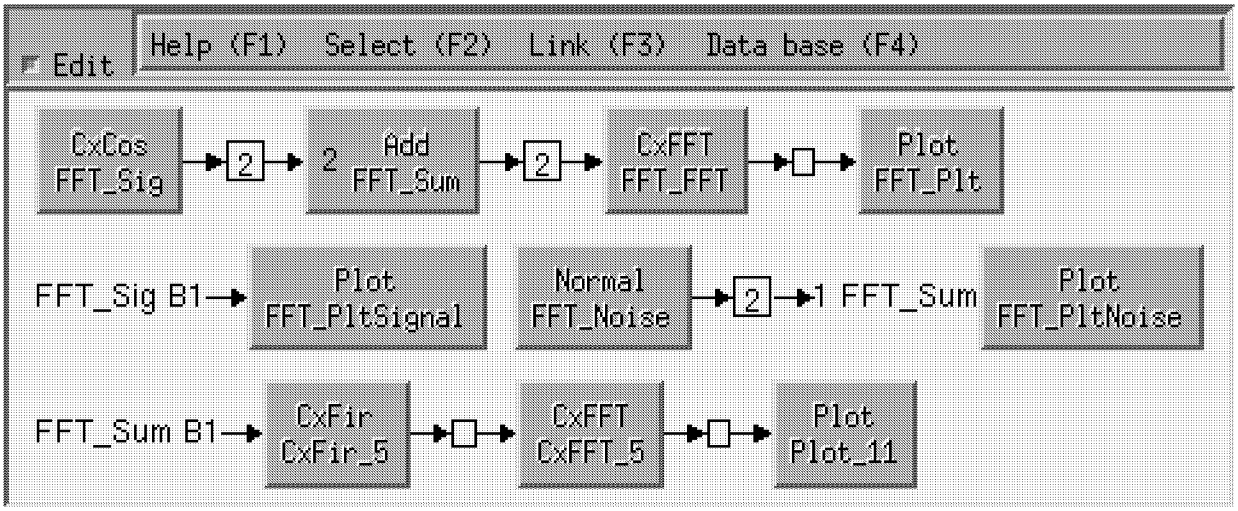
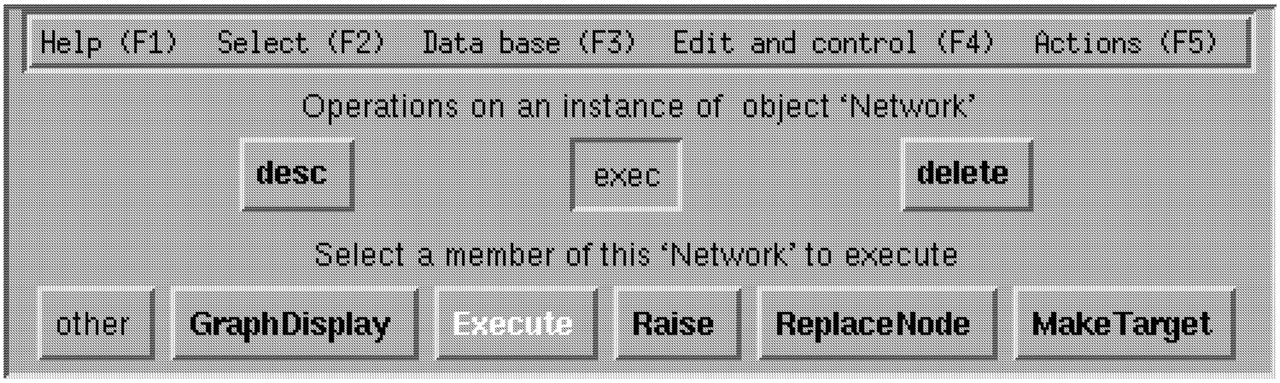


Figure 25: Completed edit of FFT_Net.

channel. If all the output channels of a node are connected you cannot select an output channel from it. This is always true if we are linking a new thread into an existing network as we are in this example. However you can always connect a new node to any buffer in a network. To link our FIR filter node into the network hold the `shift` key down and click the right mouse button on the `FFT_Sum` node. This node should be highlighted as in figure 24. To connect to the filter node hold the `shift` key down and click the left mouse button (to select the input) on the `CxFir_1` node. This node will be briefly highlighted and then the network will be redrawn with this connection. To complete your additions to the network hold the `shift` key down for all operations. (The `shift` key is the modifier that denotes network editing operations.) Click the middle mouse button on `CxFir_1` and then the left button on `CxFFT_2`. Finally click the the middle button on the `CxFFT_2` node and the right left button on the `Plot_4` node. Your network is complete and should look like figure 25

To execute the network hold the `ctrl` key down and click mouse button one over the gray area at the top of the `FFT_Net` window. A menu of commands for `FFT_net` should appear. Select `exec` from this menu and then `Execute` from the second line as in figure 26. You will be prompted for the number of times to execute the input nodes of the network. Make sure you select a



Operations on an instance of object ‘Network’

Select a member of this ‘Network’ to execute

Figure 26: Menu to execute edited network.

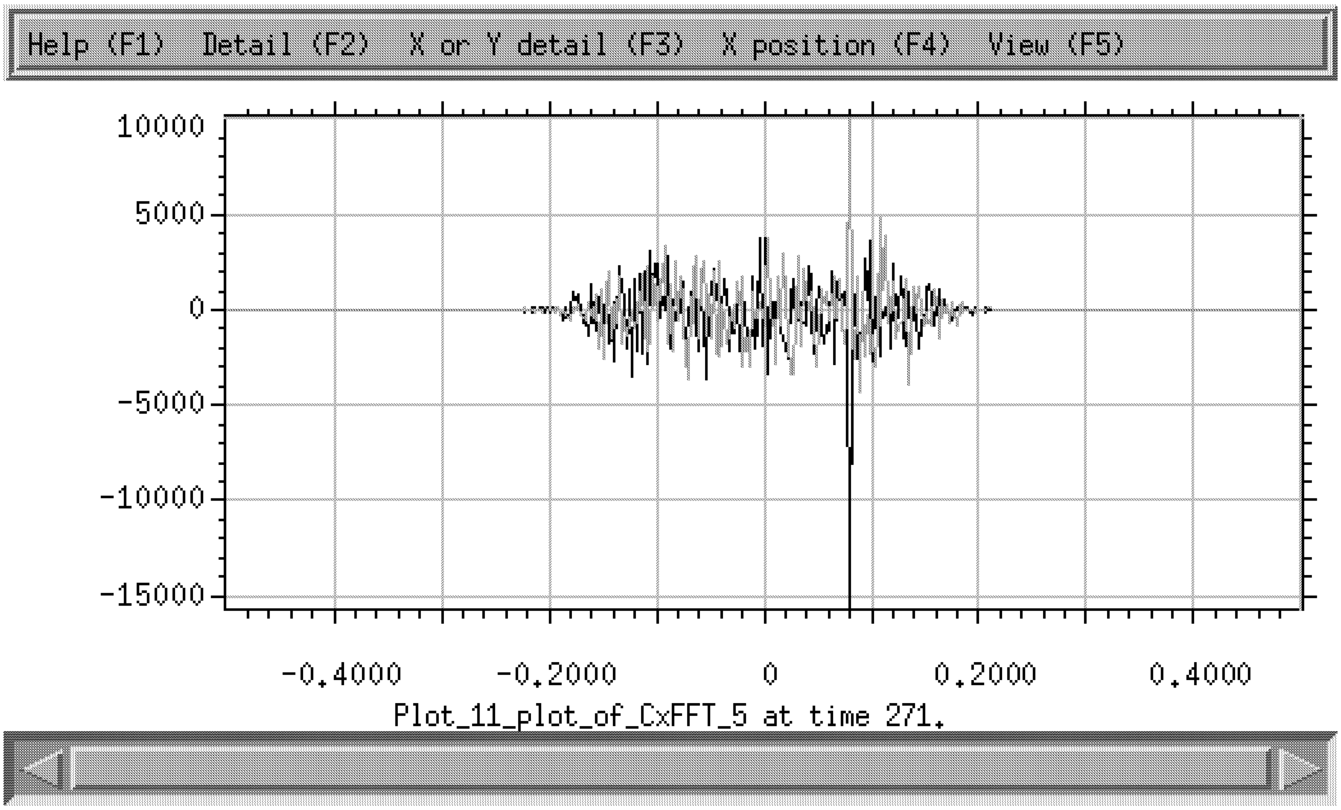
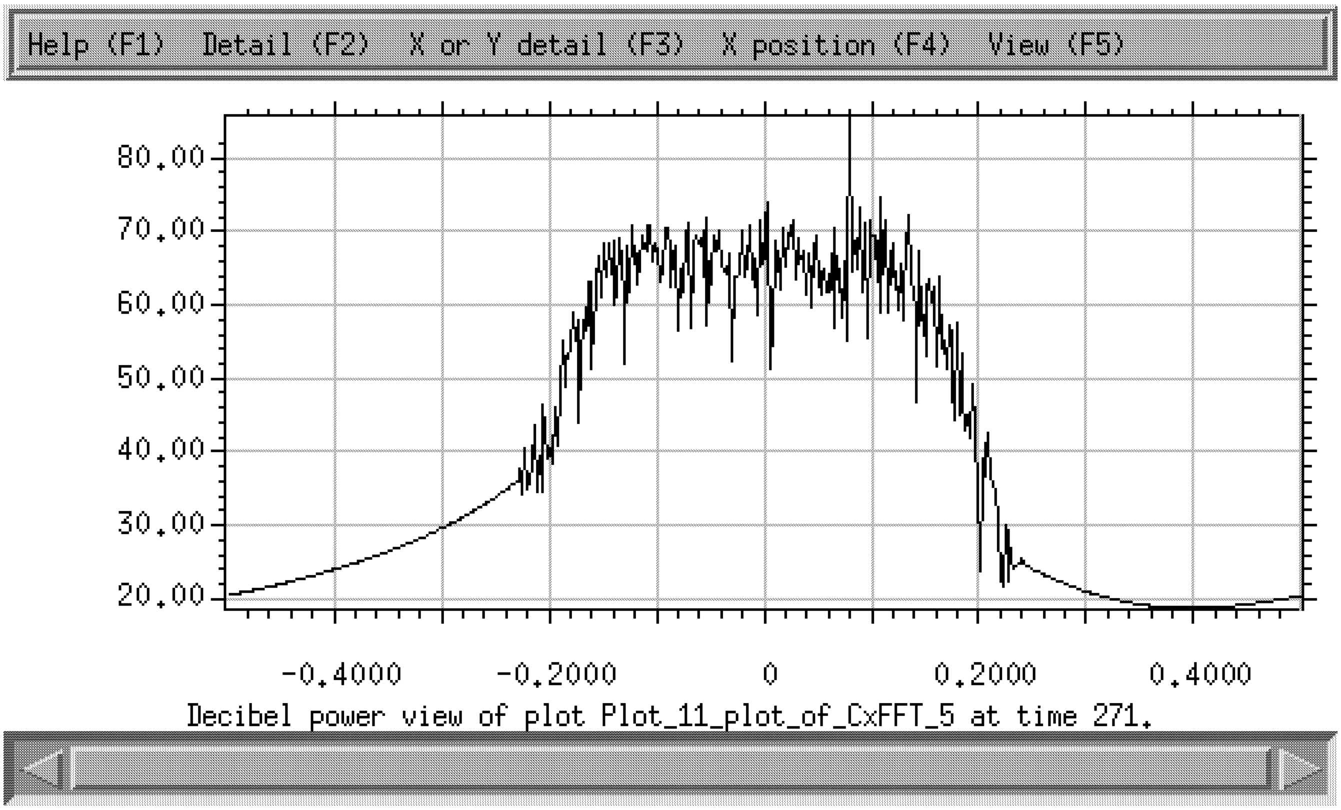


Figure 27: New complex plot from edited network.



Decibel power view of plot Plot_11_plot_of_CxFFT_5 at time 271.

Figure 28: Decibel power view of plot.

value to generate enough data for your FFT such as 1024. Execution will produce a new complex plot as in figure 27. To get a decibel power view of this data position the cursor over the plot and type `ctrl-b`. A window like that in figure 28 will appear.

You do not need to start with an existing network. If you have not checked the edit box of a displayed network (or you check it again to turn editing off) you can select a node to edit in a new network. Hold the `shift` key down while clicking the left (default parameters) or right mouse button on the selection for any DSP node in the menu data base. An instance of this node will appear in a new window. You can add additional nodes and connect them as just described and execute the network.