General Notes:

For the cursor the two screens are logically next to each other, and the cursor is on one of them at a time. Moving the cursor off the right edge of the color screen moves it onto the black&white screen, and moving it off the left edge of the B&W screen moves it onto the color screen. What the two screens display is two seperate windows into the same geometrical structure, and in general you can point to the part of the structure to which you want to refer on either screen.

Note that the bottom of the black & white screen (called the feed-back area) displays information about what the program is doing, and various parameters. Many of these parameters can be changed. As a general rule, pointing the cursor at a parameter and clicking the red button will increment the value of the parameter. The blue button will decrement it, and the yellow button allows you to type a new value (terminated by <ESC> or <CR>). A pair of exceptions are the two "scale" parameters, where the yellow button sets the scale based on the cursor position up and down the scale bar. For more details, see the parameters section, below.

Chipmonk thinks of the three main layers in the order: diffusion, poly, metal. Some commands refer to a layer by number, and in all such cases, Chipmonk consistently makes the association: 1=dif, 2=poly, 3=metal.

The mouse buttons have names, by which they will be referred to. The left-most button is called Mark, and is used for placing the "MARK" (a geometrical point reference) and for commands which, in general, change objects already on the screen (there are exceptions -- aren't there always). The middle button is called Draw, and is used for commands which draw new things on the screen (again there are exceptions). The right-most button is called Select, and is used for selecting and de-selecting objects. Selected objects are outlined (in white on the color screen).

Commands take several forms. Some consist of holding down <CTRL> while striking a key, some consist of holding down a key while pushing a mouse button, some consist of holding down <CTRL> and <TAB> and striking a key. A few consist of holding down just <TAB> and striking a key, a few requite <SHIFT> with <CTRL> and <TAB>, and a few are single key-strokes (like <ESC>).

Commands fall into four basic groups. One group is those commands which can be done in two ways. These commands have a character key associated with them, and typing <CTRL>"key" causes the command to be applied to all selected objects; whereas holding "key" down and pushing Mark (or Draw -- depending on the command) causes the command to be applied to the object pointed to. For example: <CTRL>D causes all selected objects to be deleted, while D-Draw causes the object pointed to to be deleted. Further, <CTRL>R causes the selected objects to be rotated, while R-Mark causes the pointed-to object to be rotated.

The second group is the selection commands, most of which consist of clicking the Select button while holding down some key. Exceptions are Select by itself (no key) and the area selects (q.v.) which can be started with A-Mark.

The third group are those commands of global significance, which usually require holding down <CTRL> and <TAB> while striking some key. The fourth group is "other". There aren't very many of these, and many are fairly unimportant.

There are a number of commands which wait for a string to be typed. When the program is waiting for a string, it inverts the color table, so that the color screen looks funny, and prompts on the black & white screen. When the string has been typed (terminated by <CR> or <ESC>) it puts the color screen back. If you wish to abort one of these commands any time before the <CR> or <ESC>, you can do so by hitting the "UNDO" key, which is the blank one below (and to the right of) <DEL>.

Commands:

The following discussion of commands is organized in what attempts to be the order in which one whould need to learn them. The notation "**<any>**" means any non-null combination of <CTRL>, <SHIFT>, or <TAB>. Commands in quotes and underlined refer to parameters in the feedback area at the bottom of the black & white screen. **It is strongly suggested that you run Chipmonk** (currently done by loading [Ivy]<Petit>Chipmonk.dm and then saying @pp)**, and try each of the following commands, in the order described.** Some things are much clearer if you see them happen than if you just read about them.

Drawing a few things:

1-Draw	will Draw (place on the screen) a **diffusion-to-metal contact** of minimum geometry, with its upper left corner at the location of the cursor when the command was given.
2-Draw	will Draw a **poly-to-metal contact** of minimum geometry, with its upper left corner at the location of the cursor when the command was given.
3-Draw	will Draw a **butting contact** of minimum geometry, with its upper left corner at the location of the cursor when the command was given.
X-Draw or **T-Draw**	will Draw a **Transistor** with its upper left corner at the location of the cursor when the command was given. The geometry ratio of the transistor will be that specified by the "X-Ratio" parameters in the feedback area. The length will be the length parameter times 2 lambda, and the width will be the width parameter times 2 lambda. **Note:** these parameters control the size at which the transistor is **created**. Once it is created, changing the parameters has **no effect** on it.
P-Draw	will Draw a **Pullup** with its upper left corner at the location of the cursor when the command was given. The geometry ratio of the transistor will be that specified by the "P-Ratio" parameters in the feedback area. The length will be the length parameter times 2 lambda, and the width will be the width parameter times 2 lambda.

Selection:

Select	will **Select** the object pointed to by the cursor (under the point of the arrow), and de-select all other objects. If there is more than one object under the cursor, Chipmonk attempts to select a "new" one. If you place the cursor on top of n objects and, without moving the cursor, do n selects, Chipmonk will cycle through the n objects, each being selected once.
<any>-Select	will **extend-Select** the object pointed to by the cursor, without de-selecting other objects. If there is more than one object under the cursor, Chipmonk selects a "new" one (one which is not selected). If you place the cursor on top of n objects and, without moving the cursor, do n <any>-selects, Chipmonk will select all n objects. **Note: <any>** means any non-null combination of <CTRL>, <SHIFT>, and <TAB>.
D-Select	will **De-Select** (un-select) the object pointed to by the cursor, without affecting other objects. If there is more than one selected object under the cursor, Chipmonk de-selects one of them.

Deletion:

D-Draw	will **Delete** the object pointed to by the cursor at the time the command was given. If there is more than one object under the cursor, Chipmonk will delete one of them. Deleted objects go to list of such things, and can be un-deleted, until they are flushed from the list.
<ctrl>D	will **Delete** all selected objects. Deleted objects go to list of such things, and can be un-deleted, unless they have been flushed from the list -- which happens explicitly (there is a command) or when the list gets unreasonably long.
<ctrl>U	will **Un-delete** the object or objects on the top of the list of deleted objects -- this is the object or group of selected objects deleted by the most recent un-cancelled delete. (Deleted objects are pushed onto the un-delete stack, and un-delete pops them). Un-deleted objects come back **selected**, so that if nothing was selected when the un-delete was done, <ctrl>D will un-do it. The un-delete list (stack) gets objects flushed from it when there are more than a few hundred.

Drawing Wires:

In Chipmonk, wiring is done by marking a starting point, then saying Draw ("draw wires"), at which point Chipmonk shows you a pair of "virtual" wires from the starting point to the cursor. The virtual wires follow the cursor around. When the virtual wires form a corner at the desired place, you say Draw, and the "older" virtual wire (the one from the starting point to the corner) gets made a real wire, and two new virtual wires extend from its end (the "corner") to the cursor. Each successive "Draw" will make one more virtual wire real. Wiring can be terminated in one of two ways. If the current two virtual wires will complete the circuit to the desired point (under the cursor), you can say Mark, and both wires will be made real (except if the second virtual wire is of zero length, in which case it is not made real), and wiring will be terminated. The second way out of wiring is to type <ESC> which simply terminates wiring, (stops the virtual wires from being drawn), does not make any virtual wires real, but leaves alone the real wires. There are two ways the virtual wires could be drawn -- with the first one vertical, or with the second one vertical. The initial pair is drawn such that the first (older) virtual wire is longer (at the time the command is given), so that if the cursor is at the same level as the mark, and to its right when the Draw is given, the first virtual wire will be horizontal. Each successive Draw reverses the direction of the virtual wires, so wires will alternate between vertical and horizontal (this is harder to describe than to understand). **Note:** each wire is an object, which can be selected, deleted, etc. **Note:** wires are drawn with the width specified under the appropriate layer name in the "current layer" area of the feedback area. These width parameters can be changed like other parameters (see "Feedback Commands") and can be initiallized from the "Chipmonk.Profile" file, q.v.

<ctrl>-1 or **1-Select**	Select **diffusion** as the "currently selected layer".
<ctrl>-2 or **2-Select**	Select **poly** as the "currently selected layer".
<ctrl>-3 or **3-Select**	Select **metal** as the "currently selected layer".
Mark	If you are **not drawing wires**, places the "mark" at the cursor location. The "mark" is indicated by a cursor-shape, drawn in a different color, and a little smaller, and outlined in a third color. What the colors are depends on what is under the mark. On the neutral background the mark is a purple cursor outlined in black. If you are already **drawing wires**, makes both (usually) virtual wires real, and terminates drawing wires.

Draw	If you are **not drawing wires**, Draw will start drawing wires on the currently selected layer. This means that two virtual wires will be shown from the mark to the cursor, and they will follow the cursor around, showing you where wires will go if you create them. If you are already **drawing wires**, Draw will make the "older" virtual wire real, and put virtual wires from its end to the cursor. Thus each Draw after the first creates one more real wire -- and also reverses the direction of the virtual wire corner.
<ESC>	If you are **drawing wires**, stop drawing wires. Virtual wires are deleted, but wires which have been made real are left. Thus "Draw, <ESC>" is a nop, and "Draw, Draw, <ESC>" will create one wire. <ESC> does other things -- it is a general "escape" from things you have gotten into, and is also a terminator for strings. In addition, <ESC> always forces a complete update (re-write) of both screens.
Q-Draw	If you are **not drawing wires**, Q-Draw will make one wire, from the mark to the x or y of the cursor (i.e. either horizontal or vertical). It will not put you into wiring mode. This is equvalent to "Draw, Draw, <ESC>", except that the length of the wire is slightly different. If you are already **drawing wires**, Q-Draw will make one more wire, and quit drawing wires (equivalent to "Draw, <ESC>").

A few random but necessary things:

<Ctrl><TAB>Q	Quit. Currently no attempt is made to ensure that files have been written out, etc.
<Ctrl><TAB>O	Write Output file. The current drawing is output as a .Chip file. Note that if you have pushed into a cell, it is the drawing of that cell that is output; structure above the cell will be lost if you then quit. The program forces the extension to be .Chip. If you type the null string (<CR>) when it asks for the file name, it uses the file name from the most recent Input or Output. Before writing the file, Chipmonk checks to see if it already exists, and if it does, asks for confirmation (Y/N).
<Ctrl><TAB>I	Input a .Chip file. The program forces the extension to be .Chip. The stuff read in is merged with anything already present.
<Ctrl><TAB>R	Restart. Flushes all data structure and puts you back in the state you were in when you first loaded the program (more or less). No attempt is made to ensure that files have been written out.

Moving and Copying:

In moving and copying what gets moved or copied is the set of all selected objects. The "source" reference point is the "mark", and the destination reference point is the location of the cursor when the command is given. In other words the relative offset caused by the command is the relative offset between the "mark" and the cursor.

C-Draw	Copy the selected items, with an offset such that the new stuff has the same positional relationship to the cursor as the old stuff has to the mark. The old stuff is de-selected and the new stuff is selected; the mark is moved to the cursor position.
M-Draw	Move the selected items, by an amount equal to the offset between the "mark" and the cursor. The mark is moved to the cursor position.
<any or none>-=	Move all selected items to the LEFT by one lambda..

<any or none>-	Move all selected items to the RIGHT by one lambda..
<any or none>-<LF>	Move all selected items UP by one lambda..
<any or none>-<DEL>	Move all selected items DOWN by one lambda..

Window Control:

The two screens are two independent windows into the chip-layout space. Each can be scaled and moved independently of the other.

<Space>-Mark	Move the color window so that the point under the cursor is at the center of the color screen. You may point to the desired center-point on either screen.
<Space>-Select	Move the black & white window so that the point under the cursor is at the center of the B&W screen. You may point to the desired center-point on either screen.
<Space>-Draw	This moves either screen by the amount the cursor moves between **pushing** the middle (draw) button and **releasing it**. The point under the cursor when you push the button is moved to the position of the cursor when you release the button. **Do not change screens while the button is down!**
"Scale"	This is the "Scale" parameter in the feedback area. There are two scale parameters, one for the color window and one for the black & white window. Either may be incremented (by clicking the mark button), decremented (by clicking the select button), or set to a value (by pointing the cursor at the desired position on the scale bar and clicking the draw button). See the section on the feedback window for details. An attempt is made to keep the screen centered in the same place as the scale changes.

Area Selects:

Area selects require a sequence of two commands, one to mark one corner of the (rectangular) area, and one to mark the other corner, and finish the command. This sequence can be done in two ways. The first is to use the "Start area select" command (A-Mark) to mark the first corner, and then one of the normal select commands (Select, with or without a key) to finish the command and tell what kind it is (new stuff only, extend select, or de-select). The other way is to place the "mark" at one corner, then give one of the select commands, while holding down the A key (doesn't work for de-select), to indicate that it is an area select, and its type. One advantage of the first method ("start area select" command) is that Chipmonk draws a box from the first corner to the cursor so you can see what is inside the box. The rule about what an area select refers to is (currently) that only those things are referred to whose bounding box is **fully** within the select box.

A-Mark then **Select** or **"Mark"** then **A-Select**	will **Select** the objects wholly inside the selection box (from the "mark" or A-Mark command, to the cursor when the Select is done), and de-select all other objects.
A-Mark then **<any>-Select** or **"Mark"** then **A-<any>-Select**	will **extend-Select** the objects wholly inside the selection box, without de-selecting other objects. **Note: <any>** means any non-null combination of <CTRL>, <SHIFT>, and <TAB>.
A-Mark then **D-Select**	will **De-Select** (un-select) the objects wholly inside the selection box,

without affecting other objects.

Cells:

A cell is a collection of objects which is treated as an object itself. It is usually a collection of things which performs a function, such as an input pad cell, or a register bit. (In Icarus these were called "symbols" -- a singularly bad name). Cells can contain other cells to any depth (a cell cannot contain itself). At the top level (when working on the main drawing) the objects of which a cell is made cannot be manipulated; the cell is treated as an atomic object. It is possible to "push" into a cell, and make its definition (collection of objects) the top level, and thereby modify the cell definition. This pushing can be done to any level, until there are no further levels of cells. It is also possible to "expand" a cell, which means that the cell instance is removed from the drawing, and replaced by instances of the objects in the cell's definition.

<ctrl>-C	Define a cell. The collection of all **selected** objects is turned into the definition of the cell, and those instances are removed and replaced by an instance of the cell. The program promts for a name for the cell (type a string, followed by <CR> or <ESC>). Cell names are upper case -- you can't type lower case -- so there is no confusion about case. If you type the null string (<CR> or <ESC>) the cell will have no name. This is OK, but if all instances of the cell get deleted at some point, there will be no handle for getting one, and the cell definition will fall into the great black hole that is waiting for us all.
C-Mark	Get a cell instance. The program promts for a cell name (type a string, followed by <CR> or <ESC>). Naturally this command only works for named cells. Unnamed cells can be gotten by copying. This can be a little tricky if the only instance of an unnamed cell is inside some other cell.
P-Mark	Push into a cell definition, by pointing. The drawing at the current level is saved away (pushed) and the definition of **the cell under the cursor** is made the current level. You can now modify the cell (including pushing into other cells in its definition). All commands work. In addition to adding, deleting, and moving objects, you can define cells. **Be careful**, however, not to do any I/O, as the drawing definition is hidden, and an output will output the cell definition only. When you are through, you can leave the cell definition with a "pop" (<any>-^, q.v.).
<ctrl>-P	Push into a cell definition, by selection. This is the same as P-Mark (above) except that the cell pushed into is the one selected (if more than one is selected, one of them is picked by the program). **Be careful**, however, not to do any I/O, as the drawing definition is hidden, and an output will output the cell definition only.
<shift>-P	Push into a cell definition, by name. This is the same as P-Mark (above) except that the cell pushed into is the one whose name you type in response to the prompt.
<any>-^	Pop from a cell definition. (Note: this is that same key as _, no case distinctions are made in commands). If you have made any changes to this cell, the program promts you for a decision about what to do with the changes. You have three choices (respond with a single character): You can flush the changes and go back to the orriginal definition; you can replace the old definition with the new (changed) one; or you can use the new definition to create a new cell, leaving the old cell as it was.

If you choose to make a new cell with the new definition, the program prompts for a name (no name is OK), and also **will replace certain instances of the old cell with the new one** -- specifically those instances by which you pushed into the definition. Which instances these are depends on which psuh command was used. If you pushed by pointing at an instance (P-Mark), **the instance that you pointed** at will be replaced (all others remain the old cell). If you pushed by selection (<ctrl>-P) then **all selected instances** of the old cell are replaced with the new one. If you pushed by name, **no** instances will be replaced.

E-Mark Expand a cell, by pointing. The cell pointed to is deleted and replaced by instances of each object in its definition (at the same location and orientation as when the cell was there -- so that an expand should have no electrical effect).

<ctrl>-E Expand a cell, by selection. The selected cell is expanded. Currently only one cell is expanded (this should change) so that if more than one cell is selected, only one of them is expanded.

<ctrl>-Z Display all cell names. This command currently works in a kludgey fashion. A drawing is created consisting of the cell names, and the program goes into a mode in which that drawing is displayed on the B&W screen. The windowing and scaling commands are used to move around in the drawing. If there are a lot of cells, it can take a minute to generate the drawing. To get out of the mode, type UNDO (the blank key below <DEL>). The correct scale for looking at the names is 9, but you will probably need to find them using a smaller scale. **This stuff is all going to change**. While in this mode, you can select cell names. <Ctrl>-D will then delete all selected cell definitions (usually without killing the program -- it is suggested that you save your file before doing this).

Rotation and mirroring commands:

R-Mark Rotate an object, by pointing. The object pointed to is rotated to the right by 90 degrees. This is quite fast, so rotations of 180 or 270 degrees are done with repeated 90 degree rotations. Four of these is a nop. The object is rotated in such a way that the upper left corner of the result (bounding box) is in the same place as the upper left corner was before.

<ctrl>-R Rotate selected object(s). The set of selected objects is rotated to the right by 90 degrees. The set of objects is **rotated as a unit**, maintaining their relative positions. The unit is rotated such that the resulting upper left corner of the **unit** (bounding box) is in the same place as the upper left corner of the **unit** was before. If there is only one object selected, this command has the same effect as pointing to the object and doing R-Mark. However, if more than one object is selected, it is not equivalent to doing R-Mark to each of them, as that would rotate each one in place, whereas <ctrl>-R rotates them as a unit.

M-Mark Mirror an object, by pointing. The object pointed to is mirrored (reflected) left to right (about the Y axis) and its center remains the same. Two of these is a nop.

<ctrl>-M Mirror selected object(s). The set of selected objects is mirrored (reflected) left to right (about the Y axis) **as a unit**, maintaining their relative positions. The unit is mirrored such that the center of the bounding box of the **unit** remains the same.

Width & Length commands:

Some types of objects have widths and/or lengths which are parameters. For those objects, it is possible to change those parameters with some or all of the five following commands: Widen, Narrow, Lengthen, Shorten, Default. Wires have both length and width, contacts (except butting contacts) have length, and transistors and pullups have both length and width. Cells have neither. Most of these parameters have defauls, which are parameters in the feed-back window. The "Default" command sets a parameter which has a default to that default value. The following table summarizes these parameters:

Type of Object	Width	Length	Width Default	Length Default
Wire	Y	Y	Layer/Wd	**none**
Contact	N	Y	--	4 lambda
Xstr	Y	Y	X-ratio	X-ratio
Pullup	Y	Y	P-ratio	P-ratio
Cell	N	N	--	--

L-Mark	Lengthen an object, by pointing. The object pointed to is lengthened by one lambda if it has length as a parameter (wire, contact, xstr, or pullup).
<ctrl>-L	Lengthen selected object(s). All selected objects which have length as a parameter are lengthened by one lambda.
S-Mark	Shorten an object, by pointing. The object pointed to is shortened by one lambda if it has length as a parameter (wire, contact, xstr, or pullup). There is a minimum length below which an object will not be shortened.
<ctrl>-S	Shorten selected object(s). All selected objects which have length as a parameter are shortened by one lambda. There is a minimum length below which an object will not be shortened.
W-Mark	Widen an object, by pointing. The object pointed to is widened by one lambda if it has width as a parameter (wire, xstr, or pullup).
<ctrl>-W	Widen selected object(s). All selected objects which have width as a parameter are widened by one lambda.
N-Mark	Narrow an object, by pointing. The object pointed to is narrowed by one lambda if it has width as a parameter (wire, xstr, or pullup). There is a minimum width.
<ctrl>-N	Narrow selected object(s). All selected objects which have width as a parameter are narrowed by one lambda. There is a minimum width.
0-Mark (zero)	Default width/length of an object, by pointing. The object pointed to has its width and/or length set to the default if there is a default. If there is not a default for a parameter, that parameter is not changed. This will set the width of a wire to the value in the feed-back window for the width of a wire on that layer. Wire length is unchanged. Transistors and pullups get both width and length set to the value in the X-Ratio or P-Ratio paramters. Contacts (except butting) get their length set to 4 lambda. Butting contacts are unchanged.
<ctrl>-0	Default width/length of selected object(s). All selected objects have their width and/or length set to the default if there is a default. If there is

	not a default for a parameter, that parameter is not changed.
S-Draw	Stretch a wire. This applies to all wires which are selected. Selected objects which are not wires are ignored. The "Mark" must be touching a selected wire when this command is given. The command causes one end of all selected wires to be shortened or lengthened by the difference between the "mark" and the cursor. The end which is changed is the one which the mark is nearest. For horizontal wires only the horizontal difference between the "mark" and the cursor is relevant. For vertical wires only the vertical difference is relevant. For example, if you select a group of horizontal wires and put the mark exactly on the left end of one of them, then move the curosr so it is to the left of the mark and give this command, the wire which had the mark on it will be stretched to meet the cursor, and all the other selected wires will be stretched by the same amount. If the cursor had been to the right of the mark, the wires would have been shortened. If the mark had been on the right end, that end would have been stretched/shrunk to meet the cursor. If both vertical and horizontal wires are selected, the effects will be (probably) undesirable.

Random commands:

<Ctrl><TAB>C	Write CIF file. A CIF file is made from the current drawing. Note that if you have pushed into a cell, it is the drawing of that cell that is output; structure above the cell will not appear in the CIF file. The program forces the extension to be .CIF.
<Ctrl><TAB>H	Write Hardcopy file. A press file is created containing the part of the drawing which is currently begin displayed on the **Color** screen. The program forces the extension to be .Press.
B-Mark	Make burried contact. A rectangle on the "Burr" layer is created. If the cursor is touching the intersection of a Dif line and a Poly line, the rectangle will be centered on the intersection and will be of a size and shape to meet the minimum design rules for such a contact. If the cursor is not touching such an intersection, the rectangle will be a minimun square. The rectangle can be changed in shape and size using the width and length commands.
<ctrl>-<space>	Flip wires. Reverses the direction of the corner of the temporary wires when drawing wires (q.v.).
<Ctrl><TAB><Shift>F	Flush un-delete list. When objects are deleted (q.v.) they go to a list of deleted objects, rather than being destroyed completely, so that they can be un-deleted (q.v.). This command flushes (destroys) everything on that list. That list normally gets old things flushed when it grows very big. This command allows you to recover some space.
<TAB>O	Make overglass rectangle. The programs hunts for a metal wire (rectangle) under the cursor. When it finds one, it creates an overglass rectangle centered on the metal one, and smaller in both dimensions by a standard amount (8 lambda, or 4 lambda on each edge). Once the rectangle has been made, its dimensions can be changed with the width and length commands.
T-Mark	Enter text. If the cursor is on a wire or contact, the program waits for you to type a string (terminated by <CR> or <ESC>) which it then attaches as a property to the wire or contact. The text appears on the Black&White screen. On the color screen, its position is indicated by a

small white rectangle in the upper left corner of the object. The text gets put into the CIF file, so it can be used to give a node a name (the CIF extractor finds it). The text currently does not get put into a press file written with the Hardcopy command (but will show up in one made with the CIF printer program).

<Ctrl><TAB>S	Output Spice file. The program asks for a file name, then writes a .CKT file, which can be input to spice. The file will contain transistors, and capacitors to simulate the capacitance (in addition to that associated with the gates of transistors) for each node. The generation of this information is currently both slow and n-squared. For a circuit with 50 transistors (which is probably larger than spice will accept) it takes 5 to 30 minutes to generate the file. The circuit is generated for whatever is currently at the top level, so, to keep things small, it is suggested that you gather everything you want in the spice file into a cell and push into the cell, then generate the file. Eventually a new algorithm will be used and this stuff will become faster, more accurate, and more reliable.
<Ctrl><TAB>E	Open error file. The program asks for a file name, then opens that file, ready to display its error messages (see <TAB>E below). The file should consist of error lines, each of which starts with two numbers which are the coordinates of the error.
<TAB>E	Display next error line. This causes the program to display (on top of part of the feed-back area) the next line from the file opened with <Ctrl><TAB>E (above). If the line begins with two numbers, they are taken to be the coordinates of the error in CIF units (the current setting of "Cif Lambda" is used) and the Mark is placed at that point, and that point is centered on the color screen. If the line does not begin with two numbers, the mark and screen are not moved. Then the line (without the coordinates, if any) is displayed. Note: you can go back to the first line of the file by simply re-opening it.
<TAB>F	Change font. This causes the program to change which font it uses for displaying the feedback information (and any text). There are three fonts it can use, and this cycles it through them.
<TAB>C	Change color table. This causes the program to enter a mode in which it is possible to change the settings of the current color table (which controls how the vaious layers are displayed). The command language for doing this is described below. If you get into this mode accidentally, you can get back to regular mode by typing "Q" or <UNDO>.

Color Table Control:

At the left side of the feedback window (bottom of black and white screen) is a pattern of square dots. Each dot represents a "color table". The program uses the color display in such a way that it has four bits per pixel, and it uses a particular four bit code for each layer (as well as some combinations of layers, and outlines for selection, etc.). The color table tells the display what color to use for each of the 16 four-bit codes. Consequently, changing the color table changes how the various layers (etc.) look. The dot for the color table currently selected is made larger than the others. A new color table can be selected by pointing at its dot and clicking the "Mark" button. Clicking the "Select" button anywhere in the color table area will select the default color table (upper left). The "Draw" button is used to copy color tables around and create new ones. The first click selects a table to move, and the second selects the point to move it to. If there is already a table at the destination, it is overwritten.

Color Table Editing:

<TAB>C gets you into the color table editor, editing the currently selected color table. The color screen displays a collection of objects, including 16 rectangles (one for each pixel code) at the top. It also displays three color bars at the lower right. Above the color bars is a rectangle of the currently selected color, to which the color bars refer. **Note**: color 0 is the background color and is fairly hard to see against the background. A color can be selected by pointing at an example of it in any of the objects on the screen and clicking the Select button (the invisible color-0 rectangle at the upper left is used to select the background color). The Mark button is used to move the markers up and down the color bars, and hence change the color associated with the selected pixel code. The marker will track the cursor as long as Mark is held down. If you are sloppy and cross over to another color bar, its mark will jump to the cursor. The Draw button causes the screen to be erased and re-written. When you have finished editing the color table, typing "Q" will exit the table editor.

Parameter Feedback Summary:

Following is a summary of the things in the feedback area (bottom of the black & white screen) and what they mean and how to change those that can be changed. Many of the changeable parameters are changed with a standard protocol. Such a parameter can be incremented by 1 (+1) by pointing at it and clicking the mark (left) button; can be decremented (-1) by pointing and clicking the select (right) button; can be given a new value by pointing and clicking the draw (middle) button, then typing the new value, followed by <CR> or <ESC>.

"Scale" There are two scale parameters, one for the color window and one for the black & white window. Either may be incremented (by clicking the mark button), decremented (by clicking the select button), or set to a value (by pointing the cursor at the desired position on the scale bar and clicking the draw button). The scales run from 1 to 20, 1 being too small to be useful, and 20 being ridiculously large. The particular scales are chosen to be those for which the conversion is easy, so intermediate scales are not available. When the scale is changed, an attempt is made to keep the screen centered in the same place.

"Wiring Layer:/wd" This indicates which layer is currently selected for making wires, as well as the default width for wires on each layer. The selection is indicated by a dark box drawn around the name of the layer. The default width is shown directly below the layer name, **and is in lambda**. The selection can be changed by pointing at the name of the layer and clicking any button, or by any of a number of other commands described elsewhere (see the "Drawing Wires" section). The width parameters can each be changed with the standard button protocol (mark for +1, select for -1, draw to type in a new value).

"Curs Grid" The cursor in Chipmonk is gridded, which means that it caonnot be moved to points not on the grid. This parameter is the grid point spacing, and is in **points**, each point being **1/2 lambda**. This parameter starts out being set to 2 points (1 lambda) and it is recomended that you not change it unless you really need to. It can be changed using the standard protocol, but the change does not take effect until the screen scale is changed.

"Ticks" Chipmonk puts up a grid of white dots on the color screen and this parameters controls their spacing. The parameter is in **lambda**. Setting it to 0 turns the dots off (which can save screen refresh time). It can be changed using the standard protocol.

"Size Cutoff" This is two parameters: one (labelled "B") for the black & white screen, and one (labelled "Color" on the line below) for the color screen. This parameter controls the cutoff size below which a cell is displayed as a box, instead of having all its contents displayed. Displaying as a box makes screen refresh much faster, but doesn't convey as much information. The range of values are from 1 to 200, with 1 meaning display everything and 200 meaning display only cells which fill the screen (more or less). The units are arbitrary. These parameters can be changed using the standard protocol.

"Cif Lambda" This parameter controls the lambda in 1/100th micron at which a Cif file is created. It starts out at 250, which means lambda is 2.5 micron. Since Chipmonk works in lambda, the setting of this parameter is relevant only during the actual writing of the Cif file. Several Cif files may be written from the same layout, at different scales. This parameter can be changed using the standard protocol.

"Select New" This parameter controls a mode in which newly created objects are selected, and all others de-selected. If the mode is off, creation of objects does not change what is selected.

"Orientation" This parameter controls the orientation at which new transistors and contacts are created. 0 is standard, 2 is rotated 90 degrees to the right, 4 is 180 degrees, etc. Only even values are allowed. Creating a transistor in orientation 4 is exactly equivalent to creating it in orientation 2 and rotating it once. This parameter can be changed using the standard protocol.

"Pushes" This parameter displays the number of levels deep you have pushed into cells. It starts at 0, which is the top level, and increments with each push and decrements with each pop. This parameter cannot be changed, as it displays an internal condition.

"Currently in Cell" This parameter displays the name of the cell into which you have pushed (the bottom level, if you have pushed more than one level). This is the name of the cell you are changing. This parameter cannot be changed, as it displays an internal condition.

"X-Ratio" This set of parameters (width, length, and implant) controls the default at which new (non-pullup) transistors are created. Width and length are in lambda and can be changed with the standard protocol. Implant is boolean, and can be set true with the mark button and false with the select button.

"P-Ratio" This set of parameters (width and length) controls the default at which new pullups are created. Width and length are in lambda and can be changed with the standard protocol.

"Items Selected" This parameter displays a count of the number of items selected. This parameter cannot be changed, as it displays an internal condition.

"Core Used" This parameter displays a count of the number of words currently in use by the data structure holding the drawing. There are two numbers, the first is MDS words used (used mostly for strings) and the second is "Long Pointer" space words used (for everything else). items selected. These parameters cannot be changed, as they display internal conditions.

"Mark X, Y" This displays the current location of the "mark", in lambda, in the internal coordinate system of Chipmonk. These absolute values are not of too much use (except for debugging), but differences can tell you sizes of things (see next parameter). These parameters cannot be changed, except of course by moving the "mark".

"Mark DX, DY" This displays the difference between the current location of the "mark" and its previous location, in lambda. This is useful for finding out how bit some cell or other feature (or the whole drawing) is. Placing the mark at one corner, then placing it at the opposite corner will cause the object’s size (with or without minus sign(s), depending on the corners chosen, and the order) to appear in this parameter.

New Stuff Summary:

For those who have read the previous release of this write-up, the following is a re-listing of the paragraphs which have changed since then.

Q-Draw If you are **not drawing wires**, Q-Draw will make one wire, from the mark to the x or y of the cursor (i.e. either horizontal or vertical). It will not put you into wiring mode. This is equvalent to "Draw, Draw, <ESC>", except that the length of the wire is slightly different. If you are already **drawing wires**, Q-Draw will make one more wire, and quit drawing wires (equivalent to "Draw, <ESC>").

<Ctrl><TAB>O Write Output file. The current drawing is output as a .Chip file. Note that if you have pushed into a cell, it is the drawing of that cell that is output; structure above the cell will be lost if you then quit. The program forces the extension to be .Chip. If you type the null string (<CR>) when it asks for the file name, it uses the file name from the most recent Input or Output. Before writing the file, Chipmonk checks to see if it already exists, and if it does, asks for confirmation (Y/N).

<any or none>-= Move all selected items to the LEFT by one lambda..

<any or none>- Move all selected items to the RIGHT by one lambda..

<any or none>-<LF> Move all selected items UP by one lambda..

<any or none>-<DEL> Move all selected items DOWN by one lambda..

<ctrl>-Z Display all cell names. This command currently works in a kludgey fashion. A drawing is created consisting of the cell names, and the program goes into a mode in which that drawing is displayed on the B&W screen. The windowing and scaling commands are used to move around in the drawing. If there are a lot of cells, it can take a minute to generate the drawing. To get out of the mode, type UNDO (the blank key below <DEL>). The correct scale for looking at the names is 9, but you will probably need to find them using a smaller scale. **This stuff is all going to change**. While in this mode, you can select cell names. <Ctrl>-D will then delete all selected cell definitions (usually without killing the program -- it is suggested that you save your file before doing this).

T-Mark Enter text. If the cursor is on a wire or contact, the program waits for you to type a string (terminated by <CR> or <ESC>) which it then attaches as a property to the wire or contact. The text appears on the Black&White screen. On the color screen, its position is indicated by a small white rectangle in the upper left corner of the object. The text gets put into the CIF file, so it can be used to give a node a name (the CIF extractor finds it). The text currently does not get put into a press file written with the Hardcopy command (but will show up in one made with the CIF printer program).

<Ctrl><TAB>S	Output Spice file. The program asks for a file name, then writes a .CKT file, which can be input to spice. The file will contain transistors, and capacitors to simulate the capacitance (in addition to that associated with the gates of transistors) for each node. The generation of this information is currently both slow and n-squared. For a circuit with 50 transistors (which is probably larger than spice will accept) it takes 5 to 30 minutes to generate the file. The circuit is generated for whatever is currently at the top level, so, to keep things small, it is suggested that you gather everything you want in the spice file into a cell and push into the cell, then generate the file. Eventually a new algorithm will be used and this stuff will become faster, more accurate, and more reliable.
<Ctrl><TAB>E	Open error file. The program asks for a file name, then opens that file, ready to display its error messages (see <TAB>E below). The file should consist of error lines, each of which starts with two numbers which are the coordinates of the error.
<TAB>E	Display next error line. This causes the program to display (on top of part of the feed-back area) the next line from the file opened with <Ctrl><TAB>E (above). If the line begins with two numbers, they are taken to be the coordinates of the error in CIF units (the current setting of "Cif Lambda" is used) and the Mark is placed at that point, and that point is centered on the color screen. If the line does not begin with two numbers, the mark and screen are not moved. Then the line (without the coordinates, if any) is displayed. Note: you can go back to the first line of the file by simply re-opening it.
<TAB>F	Change font. This causes the program to change which font it uses for displaying the feedback information (and any text). There are three fonts it can use, and this cycles it through them.
<TAB>C	Change color table. This causes the program to enter a mode in which it is possible to change the settings of the current color table (which controls how the vaious layers are displayed). The command language for doing this is described below. If you get into this mode accidentally, you can get back to regular mode by typing "Q" or <UNDO>.

Color Table Control:

At the left side of the feedback window (bottom of black and white screen) is a pattern of square dots. Each dot represents a "color table". The program uses the color display in such a way that it has four bits per pixel, and it uses a particular four bit code for each layer (as well as some combinations of layers, and outlines for selection, etc.). The color table tells the display what color to use for each of the 16 four-bit codes. Consequently, changing the color table changes how the various layers (etc.) look. The dot for the color table currently selected is made larger than the others. A new color table can be selected by pointing at its dot and clicking the "Mark" button. Clicking the "Select" button anywhere in the color table area will select the default color table (upper left). The "Draw" button is used to copy color tables around and create new ones. The first click selects a table to move, and the second selects the point to move it to. If there is already a table at the destination, it is overwritten.

Color Table Editing:

<TAB>C gets you into the color table editor, editing the currently selected color table. The color screen displays a collection of objects, including 16 rectangles (one for each pixel code) at the top. It also displays three color bars at the lower right. Above the color bars is a rectangle of the currently selected color, to which the color bars refer. **Note**: color 0 is the background color and is fairly hard to see against the background. A color can be selected

by pointing at an example of it in any of the objects on the screen and clicking the Select button (the invisible color-0 rectangle at the upper left is used to select the background color). The Mark button is used to move the markers up and down the color bars, and hence change the color associated with the selected pixel code. The marker will track the cursor as long as Mark is held down. If you are sloppy and cross over to another color bar, its mark will jump to the cursor. The Draw button causes the screen to be erased and re-written. When you have finished editing the color table, typing "Q" will exit the table editor.

"Select New" This parameter controls a mode in which newly created objects are selected, and all others de-selected. If the mode is off, creation of objects does not change what is selected.