Symbolic Editing with VEM

Paul B. Cohen

Massachusetts Microelectronics Center
M^2C

Contents

Tool class	Examples
UNIX	
X Window System	Version 11 (X11)
Window Manager	dxwm, uwm
Text Editor	vi, emacs, xedit
Octtools	VEM, musa, wolfe

Table 1: Tools required for the tutorials

1 Introduction

This tutorial is part of a series on the use of **VEM** and the **OCT** tool suite for VLSI design. **VEM** is an interactive graphics editor for designs represented using the **OCT** Data manager. **VEM** currently supports physical and symbolic editing of IC designs, as well as schematic capture.

Previous releases of the **OCT** tools included tutorials on physical and symbolic editing written by David Harrison, (U. C. Berkeley) author of **VEM**. In addition, a schematic capture tutorial was provided by this author. This tutorial "borrows" heavily from the earlier tutorials, but includes may new features not available in earlier versions of the **OCT** tools.

The tutorials cover many of the command available in **VEM**, but the coverage is not complete. After completing the tutorials, you should be able to discover how to use the remaining commands by referring to the manual. (Also known as the "man page".)

1.1 About the Tutorials

There are tutorials covering each of the three editing styles available in **VEM** (physical, symbolic and schematic) as of **Octtools Release 3.4**. Since each tutorial is intended to be "stand-alone", each includes a section covering some basic **VEM** operations common to the three styles. Table 1 lists the categories of tools you will need to complete these tutorials. It is assumed that you are familiar with the use of at least one text editor, the X Window System, and a window manager. If you are not familiar with the use of the X Window System and a window manager, see the tutorial, *Overview of the X Window System.*

All of the tutorials assume you are logged in to a graphics workstation running the X Window System (X11), UNIX, a window manager, and **Octtools Release 3.4** or higher. Commands you should type are noted as follows:

```
type this command
```

Quotation marks (") are very important in **VEM**. When they appear in a command such as

```
"invert:physical":open-window
```

you should enter the quotation marks as they appear in the text.

Many **VEM** commands are *bound* to a key so they can be invoked with a single keystroke. These keys will be noted with a box, as in $\boxed{\text{o}}$, and are not followed by the return key. Like UNIX, *key bindings* are case sensitive. The notation $\boxed{\text{\^{}e}}$ indicates you should press and hold the control (Ctrl) key while pressing the indicated key.

- Paragraphs like this describe an operation which is more complicated than simply typing in a command. Such operations generally require the mouse. You should read the paragraph before proceeding.

⋆ Experience has shown that there are certain operations which novice users often find difficult. These operations are noted with the symbol ⋆. Read these paragraphs especially carefully before proceeding.

Some **VEM** operations require that you *click* one of the mouse buttons, others require you to *press* a mouse button. When you are asked to *click* a mouse button, that means you should press and immediately *release* the button without moving the mouse. When you are asked to *press* one of the buttons, you should press the button and *hold* it down while performing the indicated operation (generally involving mouse movement), then *release* it when the operation is complete.

1.2 Window Managers

VEM assumes that a window manager is running. Some window managers place a title bar at the top of each window. If your terminal window has a title bar, then there is a window manager running. (Figure 1 shows the title bar created by the window manager **dxwm**.) Some window managers

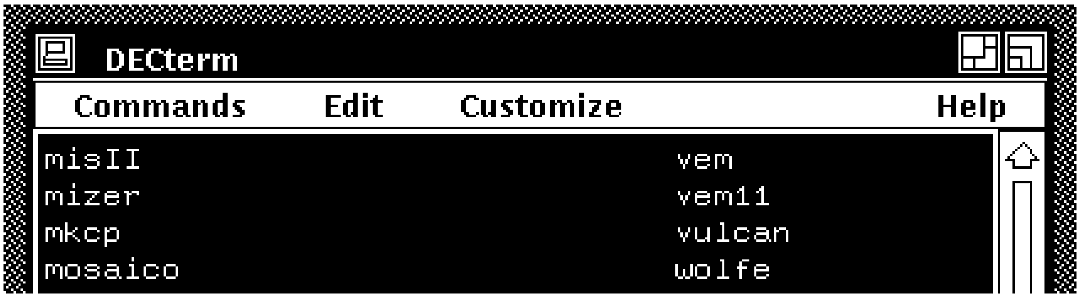

Figure 1: Sample DECterm with dxwm title bar

(like the X11 version of the popular window manager, **uwm**) do not add a title bar. If you are not sure if **uwm** is running, try moving or resizing the window. If you don't know how to do this, you should try the *Overview of the X Window System* tutorial before proceeding.

Some window managers require you to *click* one of the mouse buttons (usually the left mouse button) to *focus* keyboard input on particular window. (**Dxwm** is an example of such a window manager.) Other window managers (**uwm**, for example), *focus* input automatically, based on which window is directly under the cursor. It is assumed that you are familiar with the operation of whatever window manager you are using. The remainder of this tutorial assumes that input is focused on the proper window. Once again, if you are not sure, you should try the "Overview of the X Window System" tutorial before proceeding.

2 Starting VEM

This section explores some very basic operations in **VEM**, such as opening windows. It is very similar to the "Starting **VEM**" section in the Physical Editing and Schematic Capture tutorials. If you have completed one of these tutorials, you may safely skip to the next section after opening a **VEM** graphics window for the cell `nand:symbolic`.

- Start **VEM** with the command:

```
vem -G+0-0 &
```

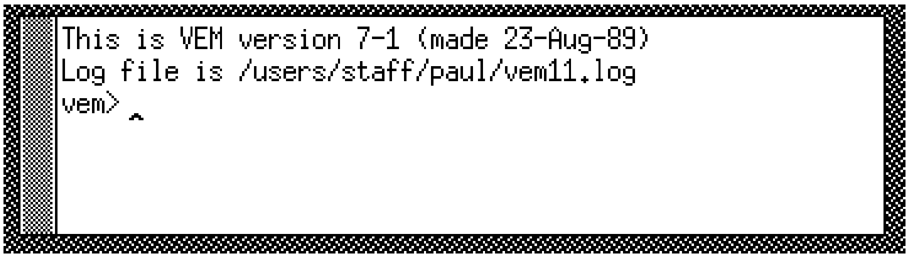

Figure 2: Sample VEM console window

(If the directory `~octtools/bin` is not included in your search path, you should either type:

`~octtools/bin/vem -G+0-0 &`

or add `~octtools/bin` to your search path.)

After a few seconds, the **VEM** console window will appear in the lower left hand corner of your screen. Figure 2 shows a sample vem console window.

The command line switch `-G+0-0` indicates the X–Y coordinates of the initial position of the **VEM** console window on the screen.

- Move the cursor to the newly created **VEM** console window. (If you need to manually focus input on a window, be sure to do so). Type:

  `"nand:symbolic"`

  and then press the $\boxed{o}$ key.

The letter "o" is the default *key binding* for the **VEM** command `open-window`. Thus, when you press the $\boxed{o}$ key, the command

`"nand:symbolic" :  open-window`

actually appears in the **VEM** console window.

- Since the `nand` cell does not yet exist, it will be created. The message: "Buffer does not exist. New buffer created" will appear in the **VEM** console window.

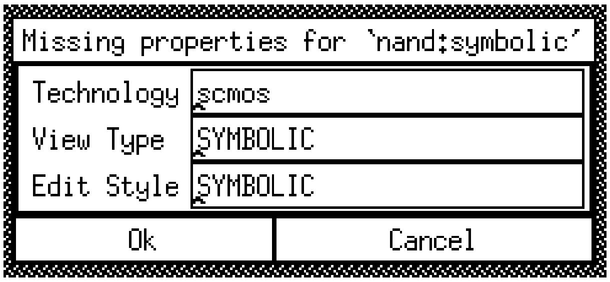

Figure 3: Sample missing property dialog box

- **VEM** will need to know a few basic facts about the cell you are creating. These facts are known as *properties*. Since the **nand** cell does not yet exist, the essential properties are considered to be missing. **VEM** will prompt you for the properties by opening a small window called a *dialog box*. (See figure 3 for a sample dialog box). **VEM** will try to choose some reasonable default values for the properties. The **Technology** property should be **scmos**, while the **EDITSTYLE** and **VIEWTYPE** properties should both be **SYMBOLIC**[1]. Move the cursor over the box labeled "OK" and *click* the left mouse button.

- The behavior of **VEM** at this point is slightly dependent on the type of window manager you are using. With some window managers, a **VEM** graphics window will be created at some location on the screen. (Be sure to change the focus!) Other window managers (**uwm**, for example) will create the outline of the **VEM** graphics window at the current cursor location. If this happens, move the cursor such that the outline is roughly centered on the screen and *click* the left mouse button to actually create the **VEM** graphics window.

Since **VEM** runs on top of the X Window System, the **VEM** graphics window and the **VEM** console window can be moved and resized in the same manner as any other X application.

Figure 4 shows a newly created **VEM** graphics window for the cell **nand**. The window is divided into two areas, the *graphics display* area and the *title bar*, located at the top of the window.

[1] If the properties are not correct, it is probably due to a typographical error when you entered the argument to the open-window command. Click the left mouse button over the Cancel box and re-enter the command line.

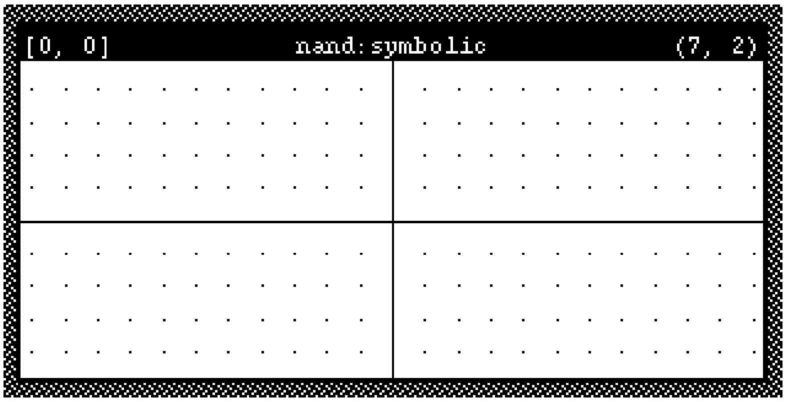

Figure 4: VEM graphics window

In the center of the title bar is the name of the cell. In this case, it is `nand:symbolic`. At the right hand edge of the title bar, **VEM** displays the current location of the cursor when it is in the graphics window, in λ (lambda).

- Move the cursor around the **VEM** graphics window and watch the numbers in the title bar change.

The numbers on the left indicate the size of a graphics argument. This will be covered shortly. For now, these numbers should be [0,0].

3 Entering Arguments in VEM

VEM commands are specified in *post-fix* notation. That is, you first specify the arguments to a command, then the command itself. For example, when you opened the graphics window, you first specified the cell `nand:symbolic`, then the command `open-window`. **VEM** accepts 5 types of arguments. These are **points**, **lines**, **boxes**, **text**, and **objects**.

Graphics arguments (points, lines, and boxes) are entered into **VEM** via the mouse. The left mouse button is used for entry of graphics information, the middle button invokes the **VEM** commands menu, and the right button is used to modify graphics arguments. As arguments and commands are entered, user feedback is provided via the **VEM** console window. Thus, it is advisable to avoid covering the **VEM** console window with other windows.

- Move the cursor to the **VEM** graphics window, and *click* the left mouse button. This will create a point in the **VEM** graphics window, and the message:

  vem> points(1)

  echos to the **VEM** console window.

- Move the cursor around the **VEM** graphics window, and enter a few more points. (Be sure to stop the cursor before entering each point.) Note that each time a point is entered in the **VEM** graphics window, the number of points in the **VEM** console window is incremented.

- Once you have entered a few points, hit the $\boxed{\text{delete}}$ key and observe that the points you entered are deleted in reverse order. The number of points in the **VEM** console window is decremented each time you delete a point.

- While you still have a few points remaining (enter a few new points if necessary), and press $\boxed{\text{\^{}u}}$. This will delete all of the points from the **VEM** graphics window, and remove the argument count from the **VEM** console window.

Next we will enter some boxes. The numbers on the left hand side of the title bar are used to indicate the size of a graphics argument. As we enter box arguments, these numbers will change.

⋆ To enter a box, *press* the left mouse button, and *hold* it down while you move the mouse. As you move the mouse, a box will appear and will change size as you move the mouse. *Release* the left mouse button to complete the box.

- The argument:

  vem> boxes(1)

  will appear in the **VEM** console window.

- Note that the argument size numbers at the left hand edge of the title bar indicate the size of the box you just drew in λ.

- Enter a few more boxes. As you draw each box, note that the argument size numbers in the title bar change as the boxes are created.

- As with points, boxes can be removed in reverse order with the delete key. Try deleting a few boxes.

- All of the arguments can be deleted by pressing ^u. Try this now.

So far we have entered graphics arguments by *clicking* the left mouse button, and by *pressing* the mouse button and *moving* the mouse before we *released* the button. This has allowed us to enter **points** and **boxes. Lines** arguments are entered into **VEM** via a combination of mouse *click* and *press* operations.

⋆ To draw a line, move the mouse to the place you would like one end of the line to be and *click* the left mouse button. *Do not move the mouse after the point has been entered.*

⋆ With the cursor over the point you just entered, *press* the left mouse button again. With the button held down, move the mouse around a bit. Note that in the symbolic editing style line arguments snap to "Manhattan" (90°) angles.

- When you want to complete the line, *release* the left mouse button.

- When you first create the point, the argument:

  vem> points(1)

  appears in the **VEM** console window. When you complete the line, the argument changes to:

  vem> lines(1)

- Try entering a few more lines and watch the response in the **VEM** console window and the argument size entry in the title bar.

- To delete a line, you will need to hit the delete key twice. The first time you hit the delete key, the last line argument in the **VEM** console window will revert to a point argument. The second time you hit the delete key, the point will be deleted. Try hitting the delete key a few times and see what happens.

- As with points and boxes, the entire argument list can be removed with ^u. If you have any lines remaining, delete them with ^u now.

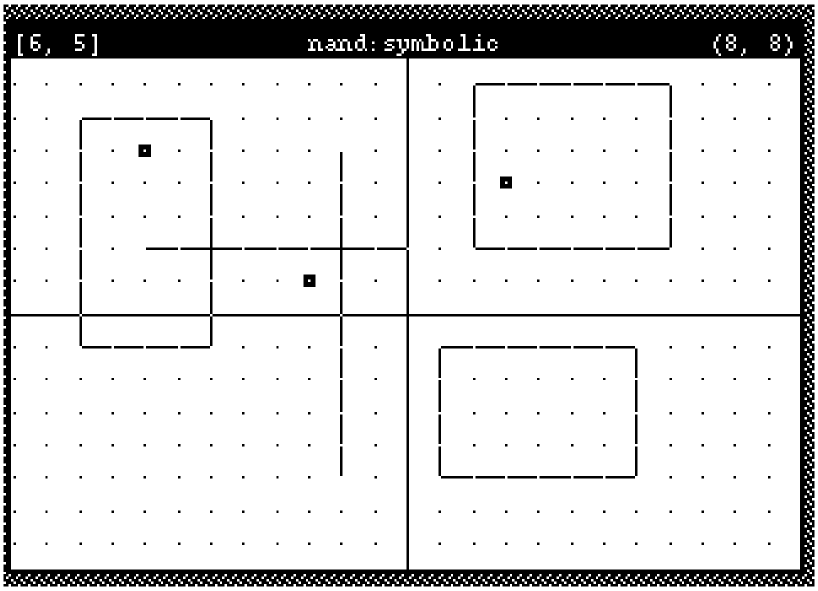

Figure 5: VEM graphics window with some points, boxes, and lines

Figure 5 shows a sample **VEM** graphics window with a some **points**, **boxes**, and **lines** entered. The argument size indicates the last argument (a **box**) drawn had a size of $6\lambda \times 5\lambda$.

3.1 Changing Arguments

Sometimes you may enter arguments which are not the right size, or are not in the right place. It is not necessary to delete an argument in order to change its position or size.

- Enter 3 or 4 points in the **VEM** graphics window. (Remember, just *click* the left mouse button to create a point.)

⋆ Move the mouse over one of the points, and *press* and *hold* the right mouse button. While *holding* the button down, move the cursor to some other location in the **VEM** graphics window and *release* the right mouse button. Note that the point has been moved to a new location.

- Delete all of the points by typing $\boxed{\hat{}\,\mathrm{u}}$.

- Enter a box somewhere in the **VEM** graphics window. (*Press* the left mouse button, move the mouse, and *release* the button.)
- Move the cursor over any of the corners of the box. *Press* and *hold* the right mouse button and move the mouse. Notice how the corner of the box moves with the cursor. *Release* the right mouse button to leave the vertex in a new location.
- Try changing the box again, noting that the argument size indicator now indicates the change in the size of the argument.
- Delete the box with with either the delete or ^u keys.
- Now enter a line somewhere in the **VEM** graphics window. (*Click* the left mouse button, then *press* the left mouse button again and move the mouse before *releasing* the button.) Position the cursor over one end of the line, and try changing the endpoint by *pressing* the right mouse button, moving the mouse, and *releasing* the button.
- Delete the line by hitting ^u.

We have seen how to enter 4 types of arguments in **VEM**. **Points**, **lines**, and **boxes** were entered via the mouse. When we first created the **VEM** graphics window `nand:symbolic`, we entered a **text** argument. We will see how to enter and manipulate **object** arguments shortly.

4 OCT Notation

VEM uses the **OCT** Data Manager to store and retrieve all graphics information. Some understanding of the **OCT** notation is useful to use **VEM** effectively.

In **OCT**, the basic design unit is the *cell*. Each cell is an arbitrarily complex portion of the design, and may include other cells in a hierarchical fashion. Figure 6 is an example of the cell hierarchy in **OCT**.

At the top of the figure is a `4 bit adder cell`. This adder includes 4 copies of a `1 bit adder cell`. Each `1 bit adder cell` might include `AND cells`, `OR cells`, and `XOR cells`. Each of these cells might include `transistor cells`, which are not shown.

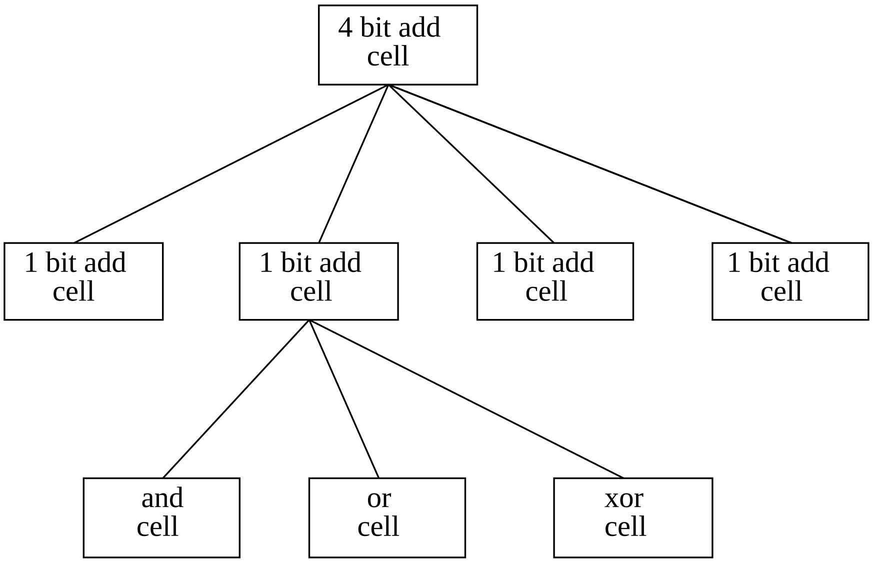

Figure 6: OCT cell hierarchy

A cell may have many *views*, depending on what design style you use and where you are in the design cycle. We have just created the `symbolic` view of the `nand` cell (though we haven't added any geometries or objects to that cell yet). The **OCT** notation for the *symbolic view* of the *nand cell* is `nand:symbolic`. We could have other views of the same cell. For example, we might also want to have a *schematic* view of the `nand` cell, where we would create the transistor level schematic. We might call this `nand:schematic`. **OCT** provides a mechanism for representing CAD data, but places no meaning on the data. *Policy* is used to assign meaning to the data. Currently, policies are defined for 3 view *types*. These are *physical*, *symbolic*, and *schematic*. The VIEWTYPE property must correspond to one of the defined types, but the name of the view is arbitrary. For example, we may have 2 different views of the `nand` cell, `nand:test1` and `nand:test2`, which both have the VIEWTYPE property **SYMBOLIC**.

Finally, each view contains some number of *facets*. Each view has a facet named "`contents`", which contains the actual definition of the view. The facet is the fundamental unit that you edit in **OCT**. An "`interface`" facet has been defined, which contains an abstraction of the contents facet. The interface facet is usually generated from the contents facet by a tool such as

vulcan.

When we opened the **VEM** graphics window we specified the **OCT** cell:view of `nand:symbolic`. Since we did not specify the facet, the `contents` facet was assumed. So when we actually add geometries to the cell, we will be adding them to `nand:symbolic:contents`.

4.1 OCT and UNIX

Each **OCT** cell corresponds to a UNIX directory, and each view is a subdirectory of the associated cell. Facets are stored as files in the corresponding view subdirectory. Thus, the **OCT** structure `nand:symbolic:contents` corresponds to the UNIX file system pathname `nand/symbolic/contents;`.

There are no restrictions on where in the UNIX file structure each of the cell directories is located. In the 4 bit adder structure of figure 6, it is reasonable to have the 4 bit adder cell and 1 bit adder cell in a directory named `project`, while the individual gates are actually in a common library.

5 Drawing Circuitry

So far, we have learned now to enter arguments, but we have not actually added any circuitry into **VEM**. We can add wiring, transistors, or other **OCT** cells using the `create` command. **Points**, **lines**, **boxes**, and **text** are used to define which **object** we would like to add and where to add it.

In **VEM** interconnect, contacts, terminals, and transistors are considered "primitives". The primitives are stored in the **OCT** library. (For scmos, the library is in `~octtools/lib/technology/scmos`). Each time we add one of these **objects** to the cell we are working on, we *create* an *instance* of that **object**. Other cells, whether they be from a common library or of our own design, can also be instantiated with the `create` command.

5.1 Palettes and the VEM Menu

In **VEM**, **points**, **lines**, and **boxes** are used to indicate <u>where</u> an instance is to be created. **VEM** will also need to know <u>what</u> you would like to create. A simple way to add objects to your cell is via *instance inheritance.* (If you have tried the physical editing tutorial, *instance inheritance* is similar

VEM	
System	⇒
Display	⇒
Options	⇒
Props & Bags	⇒
Edit	⇒
Selection	⇒
Application	⇒

Figure 7: Main VEM menu

in concept to *layer inheritance*).

When you opened the **VEM** graphics window on the cell `nand:symbolic`, **VEM** confirmed certain default property values for the cell. (See figure 3 on page 7.) The first property was the "TECHNOLOGY" property, which has a default value of **scmos**. In order to determine what layers are available for wiring in a given technology, and to create a small sample of each layer for the `create` command to use for instance inheritance, we will open a *layer palette window* for the scmos technology.

- Make sure the cursor is in the **VEM** graphics window, and *click* the middle mouse button.
- The main **VEM** menu will appear with 7 major categories, each with an arrow indicating commands specific to that category are on a sub-pane of the main menu. (See figure 7).

(If you accidentally invoke the **VEM** menu and don't want to actually execute any of the commands, simply move the cursor to some blank spot on the screen and *click* the right mouse button.)

- Move the cursor over the arrow (⇒) to the right of the **System** entry on the main menu. The System sub-menu will appear, as shown in figure 8.
- Move the cursor to the `palette` command, and *click* the left mouse button.
- A palette window will appear, with the title `.../layers:symbolic`. (Remember, you may need to position the outline and *click* the left

o	**open-window**
^d	**close-window**
?	**where**
L	**list-palettes**
P	**palette**
R	**re-read**
S	**save-window**
^S	**switch–facet**
b	**bindings**

Figure 8: System sub-pane of VEM menu

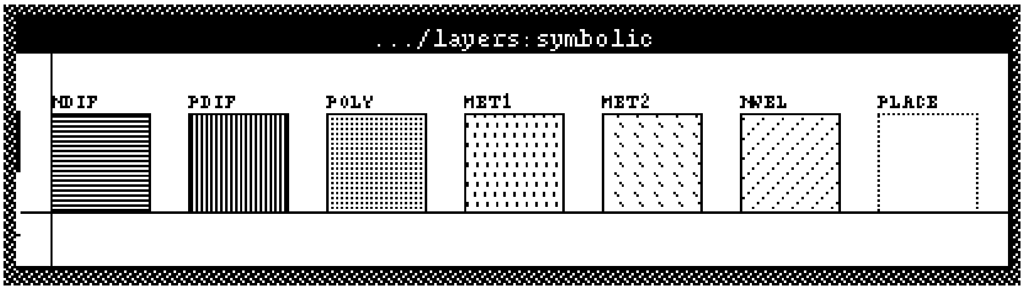

Figure 9: Symbolic editing palette for scmos

mouse button to create the window.) Figure 9 shows a sample symbolic editing layer palette for the scmos technology.

- If the layer palette window is covering the **VEM** graphics window, you should move one or both of the windows so both the layer palette window and the **VEM** graphics window are completely visible.

(The layers available in the palette do not correspond exactly to CMOS mask layers. Later in this tutorial we will use most of the layers in the palette, so we will see where each of the layers should be used.)

There are a number of other palettes available for symbolic editing. A particularly useful one is the mosfet palette, which contains a variety of PMOS and NMOS transistors.

- Be sure the cursor is in the **VEM** graphics window, and type:

  `"mosfet"`

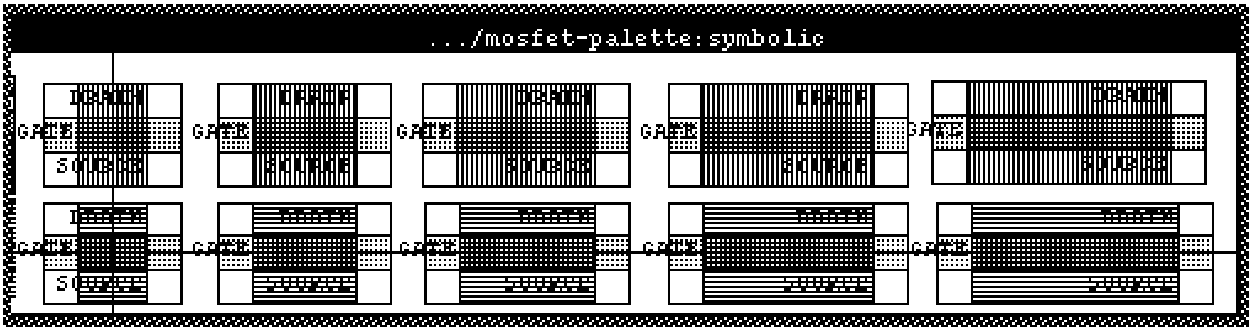

Figure 10: Mosfet Palette

Notice that the text echos in the **VEM** console window.

- With the cursor in the **VEM** graphics window, *click* the middle mouse button to invoke the **VEM** main menu again.
- Move the cursor over the arrow (⇒) to the right of the **System** entry on the main menu. The System sub-menu will appear, as shown in figure 8.
- Move the cursor to the `palette` command, and *click* the left mouse button.
- A second palette window will appear. The new palette will have the name `.../mosfet-palette:symbolic` in the title bar. The mosfet palette contains 10 transistors (5 PMOS along the top, 5 NMOS along the bottom) as shown in figure 10.
- If necessary, move the mosfet palette so it is not covering either the **VEM** graphics window or the layer palette.

There are a number of other palettes available, but the two palettes currently on the screen are the most useful. Additional transistor sizes are available on the xmosfet palette. Contacts and terminals are included on the connector palette. Since contacts and terminals are usually added automatically by **VEM**, users do not generally need to open the connector palette. For the purposes of this tutorial, we will need only the layer palette and the mosfet palette. Once you are familiar with **VEM**, feel free to explore the other palettes available.

- To find out what palettes are available, make sure the cursor is in the **VEM** graphics window, and type:

  ```
  :list-palettes
  ```

  The command will echo to the **VEM** console window, and a list of the palettes available will be printed in the **VEM** console window

5.2 Adding Transistors

In this and subsequent section, we will create the layout for a 2 input NAND gate. The gate will then be simulated and compacted. Once this is done, we will instantiate the NAND gate in another cell to demonstrate the hierarchical nature of the tools. The goal of this tutorial is to learn the use of **VEM**, not to create the best NAND gate layout possible. There may be times when you are asked to perform an operation which you suspect may not be correct. Please eperform the operations as specified, since the purpose will be to show how to correct the errors.

- In the **VEM** graphics window, move the cursor to coordinates -12,20. You can use the cursor information at the right hand edge of the title bar to determine where (-12,20) is on the screen, or you can simply count grid points.
- *Click* the left mouse buttonto create a point at (-12,20).

⋆ Move the cursor to the mosfet palette (the title bar reads `.../mosfet-palette:symbolic`) and position the cursor over the PMOS transistor in the upper left hand corner of the mosfet palette.

- *Click* the middle mouse button to invoke the **VEM** main menu.
- When the main menu appears, move the cursor to the arrow (⇒) to the right of the **Edit** entry on the **VEM** main menu.

⋆ The first item on the edit sub-pane is `create`. Move the cursor over this entry and *click* the left mouse button to create an instance of the transistor in the **VEM** graphics window. (Part of the transistor may extend off the top of the **VEM** graphics window. This is OK.)

In **OCT** terminology, you have created an instance of

`~octtools/lib/technology/scmos/primitives/pmos.4.2:physical`

in your circuit, `nand:symbolic`. The cell, `nand:symbolic` "contains" the instance of the transistor.

- Move the cursor to coordinates (0,-6) in the **VEM** graphics window.
- Create a point by *clicking* the left mouse button.
- Move the cursor to coordinates (0,-15) in the **VEM** graphics window and create another point by *clicking* the left mouse button.
- Move the cursor to the mosfet palette window, and position it over the NMOS transistor in the lower left hand corner of the palette.
- *Click* the middle mouse button to invoke the **VEM** menu, and position the cursor over the arrow ($\Rightarrow$) next to the **Edit** entry.

⋆ When the edit sub-pane appears, move the cursor over the `create` entry, *but do not click the left mouse button yet.*

- Notice the small letter "c" next to the `create` command. This is the default *key binding* for the `create` command.
- Now go ahead and *click* the left mouse button. You will see 2 NMOS transistors instantiated in the **VEM** graphics window.

The [c] key is "bound" to the `create` command. In general, there are 3 ways to enter **VEM** commands, though not all of the commands can be entered in all 3 ways. The `create` command, for example, can be entered via the menu, as we have done, be pressing the [c] key, or by typing the command preceeded by a colon(:). We will take a closer look at key bindings later in this tutorial.

5.3 Selecting and Copying Objects

While we could have instantiated 2 PMOS transistors, we can also use the `copy-objects` command to create the second transistor. Before we can copy the transistor, we will need to select it with the `select-objects` command.

- Move the cursor to coordinates (-12,20) in the **VEM** graphics window. The cursor should be centered over the PMOS transistor.
- Press the $\boxed{\text{s}}$ key to select the PMOS transistor.
- Note that the entire transistor is selected as a single object.

In the **VEM** console window, the command line indicates we have selected the fifth type of **VEM** argument, the **object**. The command line in the **VEM** console window should read:

vem> objects(1)

- *Click* the left mouse button to create a point at coordinates (-12,20).
- Move the cursor to coordinates (12,20) in the **VEM** graphics window, and *click* the left mouse button again to create a second point. The command line in the **VEM** console window will read:

  vem> objects(1) points(2)

  Thus, you are using the mouse in the **VEM** graphics window to build an argument list.
- *Click* the middle mouse button to invoke the **VEM** main menu, and move the cursor over the arrow next to the **Selection** entry.
- Locate the `copy-objects` entry on the selection sub-pane. (Note the small "x" indicating the key binding for the `copy-objects` command). *Click* the left mouse button to execute the `copy-objects` command.
- A second PMOS transistor has been instantiated in the **VEM** graphics window. This transistor is still selected. Move the cursor over the new transistor (coordinates (12,20) will work) and press the $\boxed{\text{u}}$ key, which is bound to the command `unselect-objects`. (You may have noticed earlier that `unselect-objects` can also be found on the **Selection** sub-pane of the **VEM** main menu).

The `copy-objects` command requires a list of **objects** to be copied, as well as 2 **points** indicating the displacement of the copy. **VEM** will copy the selected objects, and displace the copy by the amount indicated by the X–Y difference between the points. In this case, the points were 24λ apart in the X direction and 0λ apart in the Y direction.

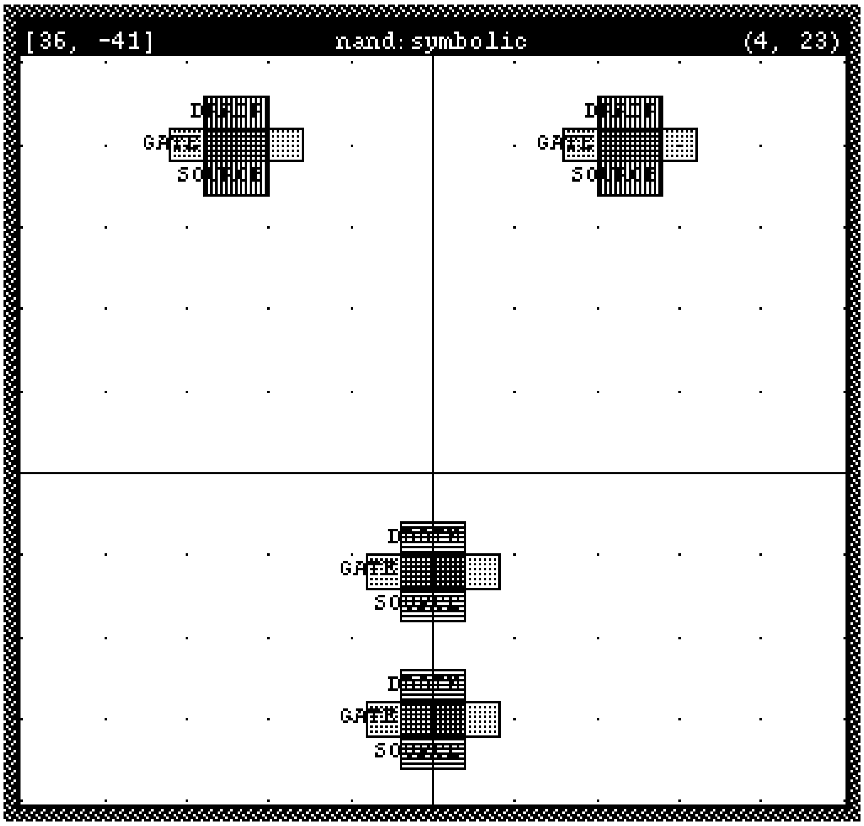

Figure 11: Partially completed NAND gate layout

The 2 PMOS transistors probably extend a bit off the top of the **VEM** graphics window. We can use the `show-all` command to fit what we have drawn so far into the **VEM** graphics window.

- With the cursor anywhere in the **VEM** graphics window, press the f key, which is the key binding for the `show-all` command. (`Show-all` is on the **Display** sub-pane of the **VEM** menu).

At this point, your layout should resemble figure 11.

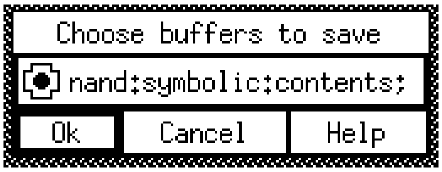

Figure 12: VEM dialog box for the save-all command

5.4 Saving Your Work

It is generally a good idea to save you work periodically. There are 2 command in **VEM** for saving you work. These are `save-window` and `save-all`. Both commands are bound to the same key, $\boxed{\text{S}}$. This is an example of the context sensitive nature of **VEM**.

- Move the cursor to the **VEM** graphics window, and press the $\boxed{\text{S}}$ key. (Remember, **VEM** is case sensitive.)
- You will see the command `:save-window` echoed in the **VEM** console window.
- Move the cursor to the **VEM** console window, and press the $\boxed{\text{S}}$ key again.
- Notice the command `save-all` is echoed in the **VEM** console window.

VEM will open a dialog box asking you which buffers you would like to save. Figure 12 shows what this dialog box will look like.

- Move the cursor over the box next to `nand:symbolic:contents;`, and *click* the left mouse button. A dot (•) will appear in the box.
- Move the cursor over the box labeled "OK", and *click* the left mouse button again to confirm the `save-all` operation.

5.5 More Selecting, Moving Transforming and Unselecting

It will make the connection of the transistors a bit simpler if the drain terminals of the PMOS transistors were "facing" each other. We can accomplish this with the `transform` command.

- In the **VEM** graphics window, move the cursor to coordinates (-17,16). This is just below and to the left of the left hand PMOS transistor.

- Draw a box exactly $10\lambda \times 8\lambda$. Remember, to draw a box you should *press* the left mouse button and move the mouse while *holding* the button. *Release* the button when the box is the right size. (Use the argument size entry in the title bar to determine the size of the box.)

- Press the $\boxed{\text{s}}$ key. This will select all objects completely enclosed by the box. In this case, the only object completely enclosed by the box is the PMOS transistor.

⋆ Keep the cursor in the **VEM** graphics window and type:

  `"270"`

  and press the $\boxed{\text{t}}$ key. ($\boxed{\text{t}}$ is the bound to the `transform` command on the **Selection** page of the **VEM** menu).

⋆ The selection box has changed, but the transistor itself hasn't moved. Press the $\boxed{\text{m}}$ key to actually move the transistor to its transformed location.

- Place the cursor over the selected transistor.

- *Click* the middle mouse buttonto invoke the **VEM** menu, and the locate the `unselect-objects` command on the **Selection** sub-pane of the **VEM** menu.

- *Click* the left mouse button to execute the `unselect-objects` command. Note that the key binding is "u".

- Move the cursor to coordinates (6,20) in the **VEM** graphics window.

- Draw a line, beginning at (6,20) and extending 8λ to the right ([8,0]). (Remember, *click* the left mouse button to create a point, then *press* the left mouse button over that point and move the mouse to the right. *Release* the left mouse button when the end of the line is where you want it.) The end of the line will be over the right hand PMOS transistor.

- Press the $\boxed{\text{s}}$ key. This will select all objects the line intersects, which in this case is only the right hand transistor.

- Press the $\boxed{\text{t}}$ key to transform the selection box.
- Press the $\boxed{\text{m}}$ key to move the transistor to its transformed location.
- With the cursor over the selected transistor, press $\boxed{\text{u}}$ to unselect the object under the cursor (the transistor).

In **VEM**, we can use **points**, **boxes**, or **lines** as arguments to the `select-objects` command. The `unselect-objects` command, and many other **VEM** commands, operate in a similar fashion, as we shall soon see.

5.6 Pan, Zoom, and Multiple Windows.

In order to complete the NAND gate, we will need to interconnect the transistors. In order to connect the source terminals of the PMOS transistor, we will need to pan the **VEM** graphics window up so we have more room above the layout to work.

- Place the cursor between the 2 PMOS transistors in the **VEM** graphics window, at coordinates (0,20).
- Press the $\boxed{\text{p}}$ key to **pan** the point under the cursor to the center of the **VEM** graphics window.

We won't be needing the mosfet palette for the remainder of this tutorial. In order to keep the screen uncluttered, we can close the mosfet palette window. We will still need the layer palette.

- Move the cursor over the mosfet-palette window, and press $\boxed{\text{\^{}d}}$. This is the key binding for `close-window`, which is on the **System** sub-pane of the **VEM** menu.

We can now begin to interconnect the NAND gate.

- Move the cursor over the drain of the left hand PMOS transistor. Coordinates (-10,20) in the **VEM** graphics window will place the cursor over the center of the drain.

- Draw a line from (-10,20) to (10,20). (*Click* the left mouse button, then *press* the left mouse button with the cursor over the point and move the mouse). The argument size entry will read [20,0].
- Move the cursor to the layer palette, and position it over the box labeled PDIF. (PDIF stands for **P DIF**fusion).
- Press the $\boxed{\text{c}}$ key to create a path on the layer PDIF.

In the symbolic editing sytle of **VEM**, connections are by default *snapped* to 90° angles. In order to make the accurate positioning of vertices simpler, the last vertex created in **VEM** remains movable until a new vertex is created or the line is terminated.

- Move the cursor to the source of the left hand PMOS transistor, at coordinates (-14,20).
- Draw a line from (-14,20) to (-20,20).

⋆ With the cursor over the end of the line at (-20,20), *press* the left mouse button and move the cursor upwards to coordinates (-20,30). This will continue the line you just created, but a vertex will be created. As you move the cursor upwards, move it left and right a bit, and notice how the vertex at moves with the cursor. *Release* the left mouse button at (-20,30)

⋆ With the cursor at coordinates (-20,30), *press* the left mouse button, move the cursor to coordinates (20,30), and *release* the left mouse button.

⋆ With the cursor at coordinates (20,30), *press* the left mouse button, move the cursor to coordinates (20,20), and *release* the left mouse button.

⋆ With the cursor at coordinates (20,20), *press* the left mouse button, move the cursor to coordinates (14,20), and *release* the left mouse button.

- Move the cursor to coordinates (0,20), and press the $\boxed{\text{c}}$ key to create a path on PDIF connecting the sources of the PMOS transistors.

Note that *instance inheritance* can be used with the cursor over any sample of the object you want to create. The various palettes are just a simple way to get a sample of various useful objects on the screen to use for instance inheritance.

At each vertex, **VEM** has automatically created a terminal.

- For the next operation, we will need to be able to see the NMOS transistors in the **VEM** graphics window, so press $\boxed{\text{f}}$ to execute the **`show-all`** command.

VEM is a multiple window application using the X Window System. This means we can have multiple **VEM** graphics windows open on a design, or we can have different designs in different **VEM** graphics windows on the screen at the same time. **VEM** allows us to work between windows on the same design.

- Move the cursor to coordinates (-7,-9) and draw a $14\lambda \times 16\lambda$ box (the argument size entry in the title bar will read [14,16]).

⋆ Press the $\boxed{\text{o}}$ key to open another **VEM** graphics window, and position it on the screen so it is not covering the the Y axis of the original **VEM** graphics window. You should leave at least 20λ on either side of the axis uncovered. (Covering the edge of the original window is OK. This is not a **VEM** restriction, but if you cover the Y axis, the next step will be hard to do). Be sure to leave the layer palette window uncovered.

⋆ Move the cursor to coordinates (-4,15) and draw an $8\lambda \times 10\lambda$ box. (If you've covered the argument size entry, the other corner of the box will be at (4,25).

⋆ Press the $\boxed{\text{o}}$ key to open a third **VEM** graphics window, and position it so it is not covering the second **VEM** graphics window. Covering the original **VEM** graphics window is OK at this point, but if you can avoid it, do so. Be sure to leave the layer palette window uncovered.

- Move the cursor to coordinates (0,-4) in the second **VEM** graphics window, and draw a 6λ vertical line. The argument size entry will be [0,6], and the endpoint will be at (0,2).

- Move the cursor to the layer palette window, and position it over the lox labeled NDIF. (NDIF stands for **N DIF**fusion).

- Press the $\boxed{\text{c}}$ key to create a path on NDIF.

⋆⋆ Position the cursor at coordinates (0,2) in the second **VEM** graphics window.

⋆ *Click* the left mouse button, and then *press* it again to begin to draw a line. Move the mouse to coordinates (0,20) in the third **VEM** graphics window, and *release* the left mouse button. (The end of the line will be centered over the PDIF path we created earlier).

- Move the mouse to the layer palette window, and position it over the box labeled MET1. Press the $\boxed{\text{c}}$ key to create a path on first metal. (MET1 stands for **MET** al **1**).

- Notice the contacts between MET1 and NDIF in the second **VEM** graphics window and between MET1 and PDIF in the third **VEM** graphics window have been added automatically.

When a newly created path terminates on an existing path ($\top$), **VEM** assumes you want the 2 paths connected, and adds the appropriate contact, if possible. When a new path crosses an existing path (+), no connection is assumed.

- Close the third **VEM** graphics window by moving the cursor into that window and pressing $\boxed{\text{\^{}d}}$ to close it. Notice the metal connection appears in the original **VEM** graphics window also.

Many other **VEM** commands also operate between windows.

- In the original **VEM** graphics window, move the cursor to a point midway between the 2 NMOS transistors. Coordinates (0,-11) will do nicely.

- *Click* the left mouse button to create a point.

- Move the cursor to the second **VEM** graphics window, and press $\boxed{\text{p}}$ to **pan** the point in the original **VEM** graphics window to the middle of the second **VEM** graphics window.

- Move the cursor down a bit until it is over the bottom NMOS transistor. (Coordinates (0,-13) in the second **VEM** graphics window).

- Beginning at coordinates (0,-13), draw a line 5λ upwards. This line should end at (0,-8).
- Move the cursor to the layer palette window, and position it over the NDIF box. Press $\boxed{\text{c}}$ to create a path on NDIF in the graphics window.

Once we connect the gate terminals of the transistors, our NAND gate will be electrically complete.

- Close the second **VEM** graphics window by moving the cursor to that window and pressing $\boxed{\hat{}\,\text{d}}$ to execute the `close-window` command.
- Move the cursor to coordinates (-3,-6) in the **VEM** graphics window, which will put it over the gate of the top NMOS transistor.
- Draw a $-4\lambda \times 0\lambda$ line.
- With the cursor at (-7,-6), *press* the left mouse button and move the cursor to coordinates (-12,17). This is over the gate of the left hand PMOS transistor.
- Move the cursor to coordinates (12,17), and draw a $0\lambda \times -32\lambda$ line (32λ downward), then continue the line with a $-9\lambda \times 0\lambda$ segment (9λ to the left). The end of the second line will be over the gate of the bottom NMOS transistor.
- Since the lines are not contiguous, **VEM** will note that you have drawn 2 lines with 2 segments each by echoing:

  vem> lines(2) lines(2)

  in the **VEM** console window.

⋆ Move the cursor over the box labeled POLY in the layer palette window. *Do not try to use instance inheritance by putting the cursor over the poly gate on the transistor.* (Remember that the transistors are objects, and instance inheritance creates a copy of the object under the cursor. If you place the cursor over the gate of the transistor and invoke `create`, **VEM** will try to create another instance of the transistor, not the POLY gate.)

- Press $\boxed{\text{c}}$ to create 2 paths on POLY, connecting the transistors.
- Save your work to this point by pressing $\boxed{\text{S}}$ with the cursor in the **VEM** graphics window.

6 Labeling Terminals

At this point, the NAND gate is electrically complete, but we should label some terminals for simulation.

- Move the cursor to coordinates (-20,30). This is over one of the terminals on the PDIF layer.

⋆⋆ *Do not create a point!* Type:

  `"Vdd"`

  and press the c key to **`create`** the label for the terminal.

The terminal at (-20,30) is known as an *actual terminal.* Before we can use this symbolic cell as part of another cell, we need to define the legal connection points. The connection points are known as *formal terminals.* The **`text`** argument to the **`create`** command indicated to **VEM** that we wanted to create a formal terminal. Only actual terminals of instances or connectors can be used as implementations of formal terminals.

- Move the cursor to coordinates (0,-17), which will put it over the source of the bottom NMOS transistor. Create a formal terminal named "GND" by typing:

  `"GND"`

  and pressing the c key.

- Move the cursor to (-12,-6) and type:

  `"a"`

  and press c to create the formal terminal "a".

- Move the cursor to (12,-15) and type:

  `"b"`

  and press c to create the formal terminal "b".

- Create a new actual terminal by moving the cursor to coordinates (0,10), *clicking* the left mouse button to create a point, and pressing the [c] key to create the terminal.

- Keep the cursor at (0,10) and type:

  `"out"`

  and press [c] to create the formal terminal "out".

6.1 Adding Properties

We can add various *properties* to the NAND gate with the **`edit-property`** command.

- Move the cursor to coordinates (0,30) and invoke the **`select-net`** command by pressing [^n]. Note that the entire path on PDIF is selected, as are the source terminals of the PMOS transistors.

- Press [^e] to invoke the **`edit-property`** command. This will bring up a dialog box labeled "Editing New Property". Move the cursor to the dialog box, and position it in the area next to the word "Name". Type:

  `NETTYPE`

- Keep the cursor in the dialog box, but move it to the large box next to the word "Value". Type:

  `SUPPLY`

- Make sure the property type (along the left edge of the dialog box) is "String".

- Move the cursor over the box along the bottom of the dialog box marked "Save, and *click* the left mouse button.

- Move the cursor to coordinates (0,-17) and invoke the **`select-net`** command by pressing [^n]. This will select only the source of the bottom NMOS transistor.

- Press $\boxed{\hat{}\text{e}}$ to invoke the **`edit-property`** command. This will bring up a dialog box labeled "Editing New Property". Move the cursor to the dialog box, and position it in the area next to the word "Name". Type:

  `NETTYPE`

- Keep the cursor in the dialog box, but move it to the large box next to the word "Value". Type:

  `GROUND`

- Make sure the property type (along the left edge of the dialog box) is "String".

- Move the cursor over the box along the bottom of the dialog box marked "Save, and *click* the left mouse button.

7 Wrapping up!

Congratulations on the completion of the NAND gate layout. If everything went well, your NAND gate should look something like figure 13.

7.1 Key Bindings

As you become more familiar with **VEM**, you will find key bindings an effective way so speed up the layout process. There is a special command, **`bindings`**, which allows you to determine some information, including the key binding, for each command.

- *Click* the middle mouse button to invoke the **VEM** main menu, locate the **`bindings`** command at the bottom of the **System** sub-pane.

- *Click* the left mouse button to select the **`bindings`** command.

- In the **VEM** console window, you will see the command **`:bindings`** echoed, followed by the text:

  `Specify a command using a single key, a menu, or by type-in:`

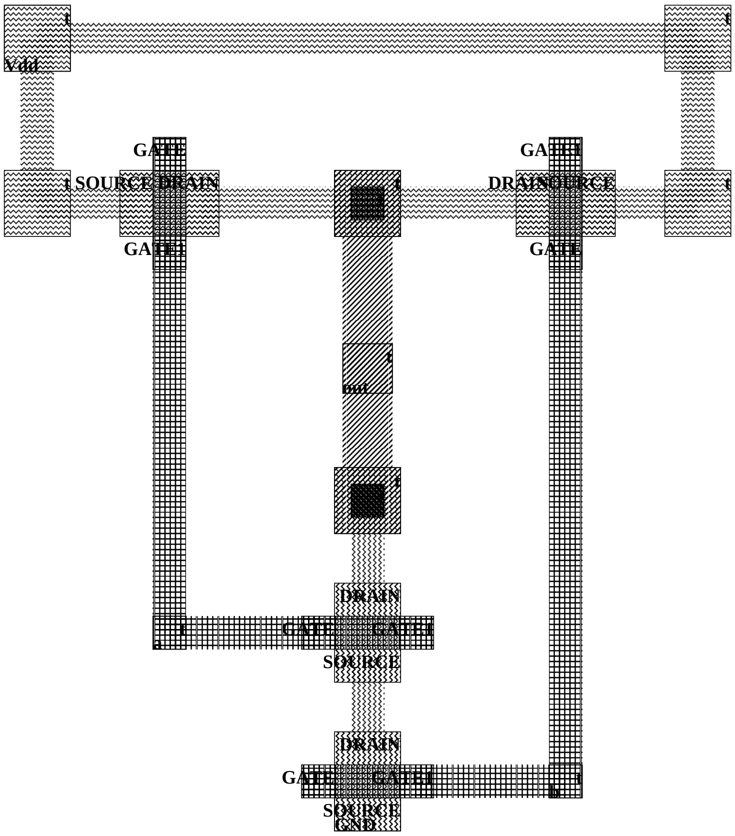

Figure 13: NAND gate symbolic layout

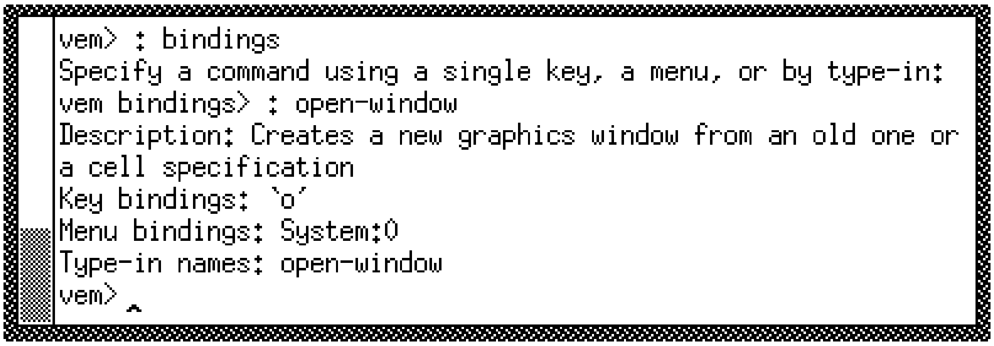

Figure 14: VEM bindings command

- You will also notice that the **VEM** console window prompt now reads:

  vem bindings>

- The next command you enter will not actually execute, but will be described by the bindings command. Press the $\boxed{\text{o}}$ key.

- In the **VEM** console window, a one line description of the command (`:open-window`) will appear, along with the key binding, and the sub-pane of the menu the command appears on. Figure 14 is a sample of the **VEM** console window after the **`bindings`** command has been executed. The notation "System:0" indicates it is the top item on the **System** sub-pane of the **VEM** main menu.

- After the command is described, **VEM** returns to normal command entry.

7.2 Closing VEM

Before you close **VEM**, you should save your work.

- With the cursor in the **VEM** graphics window, press $\boxed{\text{S}}$ to save your work.

- Move the cursor to the **VEM** console window and press $\boxed{\text{\^{}d}}$ to close **VEM**. Closing the **VEM** console window will close all of the other **VEM** windows, but first **VEM** will confirm this operation with another dialog box, shown in figure 15

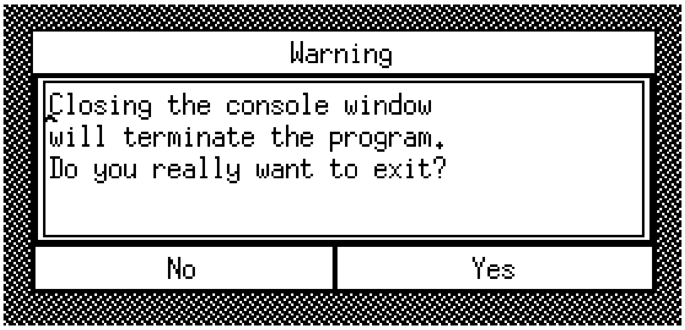

Figure 15: VEM exit warning

- *Click* the left mouse button over the box labeled "Yes" to confirm the operation.

8 Simulation

We can use the **MU**lti Level **S**imul**A**tor, **musa** to simulate the NAND gate and be sure it is correct. We will use a terminal window to run the simulator.

- From the terminal window, type:

  ```
  musa nand:symbolic
  ```

  Musa will respond with a version message, and the prompt:

  musa>

  to indicate you have invoked the simulator.

- The `set` command is used to set a signal (a formal terminal) to a value. Set "a" to 0 by typing, at the musa> prompt,

  ```
  set a 0
  ```

- The `set` can be abbreviated "se". Set "b" to 0 by typing:

  ```
  se b 0
  ```

- We do not need to set V_{DD} and GND, since **musa** recognizes the 2 NETTYPE properties, SUPPLY and GROUND, that we assigned earlier. If we had not assigned these properties to any net, then we would need to set V_{DD} and GND explicitly, as we do all other inputs.

⋆ **Musa** does not actually run the simulation until we enter the "evaluate" command. This allows us to set a number of inputs before actually running a simulation. Since both inputs have been set, we can simulate the circuit. Type:

```
evaluate
```

- The `show` command is used to inform **musa** which signals we would like to see. Type:

```
show a b out
```

- **Musa** will respond with:

```
 a:O b:O out:1
```

Musa uses the letter "O" and the number "0" to distinguish between nodes which have been set low by the user (letter "O") and nodes which evaluate to a low value (number "0"). The letter "I" and the number "1" are used to indicate whether a high value has been set by the user or is a simulation result.

- We can save some typing by setting a "watch list" in **musa**. Type:

```
watch io a b out
```

(The `watch` command can be abbreviated `wa`). Continue the simulation by typing the following sequence of commands:

```
se a 1
ev
sh io
```

In this case the `show` command (abbreviated `sh`), was used with the name of the watch list ("io") as an argument. We could have multiple watch lists.

- Finish the simulation by typing:

```
se b 1
se a 0
ev
sh io
se a 1
ev
sh io
quit
```

A complete simuation of the NAND gate is shown below.

```
musa> set a 0
musa> se b 0
musa> evaluate
musa> show a b out
 a:0 b:0 out:1
musa> watch io a b out
musa> se a 1
musa> ev
musa> sh io
 a:I b:0 out:1
musa> se b 1
musa> se a 0
musa> ev
musa> sh io
 a:0 b:I out:1
musa> se a 1
musa> ev
musa> sh io
 a:I b:I out:0
musa> quit
```

9 Compacting the Layout

When we created the layout for the NAND gate, we did not pay particular attention to the layout rules or to trying to arrive at a reasonably dense layout. **Sparcs** is a layout compaction tool which will minimize the area of the layout and satisfy the spacing rules. (If you have violated a spacing rule, **sparcs** will actually increase the area of your circuit to satsify the rule).

Many of the tools in the **OCT** suite can also be executed from within **VEM** as an RPC application. (RPC stands for **R**emote **P**rocedure **C**all). **Sparcs** is one such tool which has an "rpc" version, **rpc-sparcs**. We will use **rpc-sparcs** to compact the NAND gate so we can see what the compactor is doing.

- Start **VEM** and open a graphics window with the command:

  `vem -G+0-0 -Fnand:symbolic -G+100+100 &`

  After a while, both the **VEM** console window and the **VEM** graphics window will appear.

- *Click* the middle mouse button to invoke the **VEM** main menu, and move the cursor to the **Application** sub-pane.

- *Click* the left mouse button over the entry "`sparcs`".

- You will see a message in the **VEM** console window:

  `~octtools/bin/rpc/rpc-sparcs running on host <hostname>`

- After a few more seconds, you will see another message appear in the **VEM** console window:

  `sparcs version 1-10 (Aug 23, 1989) initialized.`

  (The date and version may be different.)

Rpc applications often have their own menus, which are invoked by holding down the Shift key while *clicking* the middle mouse button.

- Press the Shift key and *click* the middle mouse button to invoke the **rpc-sparcs** menu.

- Move the cursor over the arrow (⇒) to the right of the **options** entry on the **rpc-sparcs** main menu. This wil invoke a sub-pane.

- On the **options** sub-pane of the **rpc-sparcs** menu, move the cursor over the `show-each-iteration` entry. (You do not need to hold the Shift key). *Click* the left mouse button. The command will echo in the **VEM** console window. (This option will cause the compacted layout to be re-drawn after each iteration of the compactor. It is not really necessary to do this, but it is fun to watch).

- While holding the Shift key, *click* the middle mouse button to invoke the **rpc-sparcs** menu again.

- On the **run** sub-pane, *click* the left mouse button over the `RUN` command.

- After a few seconds, another **VEM** graphics windowwill open, with what appears to be a copy of the NAND gate. (It is a copy of the NAND gate). The title bar for this new window indicates the name of the cell is `nand:spaced`.

- In a little while, you will see the layout be compacted, first in the horizontal direction, then in the vertical direction. It may take more than one iteration to compact the layout.

- When **rpc-sparcs** can no longer decrease the area of the layout, the message:

  `Converged`

  will appear in the **VEM** console window.

- With the cursor in the original **VEM** graphics window(`nand:symbolic`), hold the Shift key and *click* the middle mouse button. When the **rpc-sparcs** menu appears, move the cursor to the **run** sub-pane and *click* the left mouse button over the `exit-sparcs` entry.

For the next section we will need **VEM**, so we should not close the **VEM** console window. In order to keep the screen uncluttered, we should close the 2 **VEM** graphics windows.

- Move the cursor to either of the **VEM** graphics windows, and press ^d to close it.

- Move the cursor to the remaining **VEM** graphics window, and press ^d to close it.

10 Creating the Interface Facet and Adding Properties

Before we can instantiate the NAND gate in another cell, we will need to create the interface facet. **Vulcan** will create the interface facet.

- From the terminal window, enter the command:

  ```
  vulcan -k nand:spaced
  ```

  We should also add the "CELLTYPE" property to the interface facet. The value of the property should be "MODULE". We can use **octprop** to do this.

  - From the terminal window, type:

    ```
    octprop -pCELLTYPE=MODULE nand:spaced:interface
    ```

11 Instantiating One Cell in Another

With the NAND gate complete, we can instantiate it in another cell as often as we wish.

- From the **VEM** console window, type:

  ```
  "ao1:symbolic"o
  ```

- Since the cell `ao1:symbolic` does not exist, **VEM** will open a missing property dialog box, The correct property values should be:

Technology:	scmos
View Type:	SYMBOLIC
Edit Style:	SYMBOLIC

- If the dialog box is correct, *click* the left mouse button over the "OK" box. (Remember, you may need to position the **VEM** graphics window and *click* the left mouse button to actually create the **VEM** graphics window).

We will use 2 instances of the compacted NAND gate to create a circuit with the logic function:

$$output = (in1 \cdot in2) + \overline{in3}$$

which will be implemented using 2 input NAND gates as:

$$output = \overline{\overline{(in1 \cdot in2)} \cdot in3}$$

- Move the cursor to the newly created, blank **VEM** graphics window. Position it at coordinates (0,0).
- *Click* the left mouse button to create a point.
- Type:

  `"nand:spaced"`

  and press $\boxed{\text{c}}$ to create an instance of the NAND gate.
- The NAND gate will probably not be centered in the **VEM** graphics window. Press $\boxed{\text{f}}$ to execute the **`show-all`** command.
- With the cursor anywhere over the NAND gate, press $\boxed{\text{s}}$ to select the gate. Notice that the entire NAND gate is selected as a single entity. The command line in the **VEM** console window reads:

  vem> objects(1)
- Press $\boxed{\text{Z}}$ to **`zoom-out`**. The design will appear smaller.
- Move the cursor to coordinates (-25,0) and *click* the left mouse buttonto create a point.
- Create another point at (5,0) by *clicking* the left mouse button at coordinates (5,0).
- Execute the `copy-objects` command by pressing $\boxed{\text{x}}$. (If you prefer to use the menu, feel free to do so).
- Press $\boxed{\text{f}}$ again to fit everything inside the **VEM** graphics window.
- Press $\boxed{\hat{\ }\text{u}}$ to unselect everything.

We have created 2 instances of the NAND gate in our cell ao1. We can connect them to form the new function. For the purposes of this tutorial we will keep the number of open windows to a minimum, even though this may not be the most efficient way to proceed. Once you are familiar with **VEM**, you should feel free to experiment to find a style which is comfortable for you.

- In the **VEM** graphics window, create a $15\lambda \times 6\lambda$ box beginning at (-3,12) and ending at (12,18).
- Press $\boxed{\text{z}}$ to `zoom-in` such that the box now occupies the entire **VEM** graphics window.
- Press $\boxed{\text{P}}$ top open a layer palette window. This is the only palette you will need for this circuit.
- Draw an 11λ line from (-1,15) to (10,15).

⋆ Move the cursor to the PDIF box in the layer palette window, and press $\boxed{\text{c}}$ to create the path on PDIF. (*Do not try to use instance inheritance with the cursor over the PDIF paths in the* **VEM** *graphics window*).

If you try to use instance inheritance by placing the cursor over one of the PDIF paths in the NAND gate, the results won't be quite what you expect. When you invoke the `create` command, **VEM** will try to create an instance of whatever is under the cursor. Remember when you selected to NAND gate to copy it, the entire NAND gate was selected, regardless of where (over the gate) the cursor was. So, while it may <u>look</u> like the cursor is over a path on PDIF, it is really over an instance of a NAND gate.

- Press $\boxed{\text{f}}$ (from the **VEM** graphics window) to restore the center the full layout in the window.
- Move the cursor to coordinates -10,-17 and draw a line extending 5λ downward [0,-5] to coordinates (-10,-22).
- Continue the line (by *pressing* the left mouse button) 30λ to the right [30,0]. The end will be at (20,-22).
- Continue the line 5λupward to coordinates (20,-17).

- Move the cursor to the layer palette window, and position it over the NDIF box. Press $\boxed{\text{c}}$ to create a path on the NDIF layer in the **VEM** graphics window.
- Move the cursor to (-10,0) and draw a line 15λ to the right, ending at (5,0).
- Move the cursor to the MET1 box in the layer palette window, and press $\boxed{\text{c}}$ to create a path on 1^{st} metal.
- Move the cursor back to (5,0), and draw a line 10λdownwards, to (5,-10).
- Continue the line to the right to coordinates (14,-10). The new segment will be 9λ long.
- Move the cursor over the POLY box in the layer palette window, and press $\boxed{\text{c}}$.
- Draw a 10λ line beginning at (20,0) and ending at (30,0). Create a path on 1^{st} metal by moving the cursor to the MET1 box in the layer palette and presing $\boxed{\text{c}}$.
- Draw a line from (-16,-10) extending 5λ to the left.
- Draw a line from (-5,-15) extending 5λ to the right.
- Draw a line from (25,-15) extending 5λ to the right.
- Create 3 paths on POLY by moving the cursor over the POLY box in the palette window and pressing $\boxed{\text{c}}$.

11.1 Label the I/O

- Move the cursor to coordinates (-21,-10) and type:

  `"in1"`

  and press $\boxed{\text{c}}$ to create the formal terminal named “in1”.
- Move the cursor to coordinates (0,-15) and type:

  `"in2"`

and press $\boxed{\text{c}}$ to create the formal terminal named "in2".

- Move the cursor to coordinates (30,-15) and type:

  ```
  "in3"
  ```

  and press $\boxed{\text{c}}$ to create the formal terminal named "in3".

- Move the cursor to coordinates (30,0) and type:

  ```
  "output"
  ```

  and press $\boxed{\text{c}}$ to create the formal terminal named "output".

We're ready to simulate the cell, be first we should save you work.

- With the cursor in the **VEM** graphics window, press $\boxed{\text{S}}$ to save the cell.
- Move the cursor to the **VEM** console window and press $\boxed{\text{\^{}d}}$ to close **VEM**.

12 Simulating the New Cell

We are now ready to simulate the new cell. Instead of entering the simulation commands directly into **musa**, we can create a file with the simulation commands.

- Using a text editor, create a file `ao1.sim` with the following contents::

  ```
  se in1 0
  se in2 0
  se in3 0
  ev
  wa io in1 in2 in3 output
  sh io
  se in3 1
  ev
  sh io
  ```

```
se in1 1
ev
sh io
se in2 1
ev
sh io
quit
```

- When the file is complete, enter the command:

  ```
  musa -i ao1.sim ao1:symbolic
  ```

  to simulate the circuit.

12.1 Printing the Layout (Optional)

If you have a PostScript printer, you can print the NAND gate layout. You will need to determine the name of the printer.

- With the cursor in the terminal window, enter the command:

  ```
  oct2ps ao1:symbolic | lpr -Ppostscript_printer
  ```

  where you substitute the name of your PostScript printer in the lpr command.

13 Additional Reading

There are several documents which may be of interest to the **VEM** user. In general, users should read the "man page" for each of the tools in the tutorials. There are some additional documents which may also be of interest. Some of these documents are:

David S. Harrison. *Customizing VEM.* University of California, Berkeley, 1988.
(`~octtools/document/PostScript/Vemcustom.ps.Z`)

David S. Harrison. *VEM Manual Page.* University of California, Berkeley, 1988.
(`~octtools/man/man1/vem.1`)

David S. Harrison, Peter Moore, Rick L. Spickelmier, and A. Richard Newton. *Data management an Graphics Editing in the Berkeley Design Environment.* Proceedings of ICCAD, pages 20–24, November 1986.

Rick L. Spickelmier editor. *Policy Guides for OCT Tools Distribution 3.0.* OCT Tools Distribution 3.0. University of California, Berkeley. March, 1989.
(`~octtools/document/PostScript/policy.ps.Z`)

Rick L. Spickelmier. *Remote Procedure Call Package.* University of California, Berkeley, 1986.
(`~octtools/document/PostScript/rpc.ps.Z`)