*DAToolsIntroductionDoc.tioga*
*Bland, May 12, 1987 11:51:25 am PDT*

# An Introduction to the DATools:

## Schematics, Extraction and Simulation During Your First Week in CSL

Lissy Bland



**Abstract:** This is an introduction to the design definition, simulation and layout tools available in CSL. The purpose is to give new members of the Design and Architecture Group a guided tour of the most central tools. The tour includes: ChipNDale, a graphical editor for VLSI layout and schematics, the Standard Cell Library, Extract (the layout and schematics extractor), Rosemary (a logic simulator), Thyme and SPICE (timing simulators), and the generation of layout by PatchWorkCore (PWCore).

**Created by:** Lissy Bland

**Maintained by:** Lissy Bland

**Keywords:** schematics, extraction, simulation, standard cells, design automation, ChipNDale, Sisyph, DATools, Rosemary, Thyme, Oracle, layout, VLSI, CMOSB

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

**For Internal Xerox Use Only**

## 1.0 Introduction

This is an introduction to the design definition, simulation and layout tools available in CSL. Its purpose is to give new members Design and Architecture Group a guided tour of the most central tools. It does not intend to be comprehensive. When there are several ways to achieve the same result, one good way will be presented.

This document assumes a functional knowledge of the Cedar environment. You should get someone to give you a couple lessons or read enough from the following three documents to be able to get around in the Cedar world. (You'll get up to speed faster with private lessons.)

1. [Cedar]<Cedar7.0>Documentation>TiogaDoc.tioga

   This explains how to to use the tioga text editor and get around tioga documents.

2. [Cedar]<CedarChest7.0>Documentation>Introduction.tioga

   This is a kind-of operator's manual for acquiring and using Cedar.

3. [Cedar]<CedarChest7.0>Documentation>BriefingBlurb.tioga

   This is a general introduction to the computing environment at PARC slanted towards the needs and interests of newcomers to the Computer Science Laboratory. It's aging but still quite helpful.

## 2.0 Getting Started

The very first time the datools are started on a particular machine the following sequence of commands must be executed in a CommandTool:

cd ///users/yourSurname.pa/aSubdirectory/

Using a subdirectory is useful when changing versions. All the files in that subdirectory can be deleted without disturbing files in other directories.

Bringover -p /DATools/DATools7.0/Top/BringDATools.df

Using the -p switch retrieves only the public files.

BringDATools

DAUser

Executing the above sequence of commands takes a while. Go get a cup of coffee or read your mail. About mail, most users of the DATools find that they do not have enough GFI's (Global Frame Indices) to use the DATools and the Walnut mail system. Users that want to be able to read new mail throughout the day while using the DATools should probably run the no-frills mail system which is called Peanut. (c.f. [Cedar]<CedarChest7.0>Documentation>PeanutDoc.tioga.)

Watch your mail for changes to the DATools. You should re-execute BringDATools every time there is a significant change.

For more information about the bringDATools command see, [DATools]<DATools7.0>BringDATools>BringDATools Doc.tioga. See [DATools]<DATools7.0>DAUser>DAUserDoc.tioga for additional information about the DAUser subdirectory. This subdirectory is used for commands, programs and documentation that are at the crossroads of different tools.

### 2.1 Using this Document

The first three sections of this document present basic material about getting started, using the Standard Cell Library, and drawing schematics. Section 4.0 begins with a discussion of extraction, and introduces the most basic vocabulary of Core, the data structure used by all of the DATools to describe electrical circuits. In Section 4.2 a four-bit adder is drawn to demonstrate some of the more advanced features available for schematic entry. These first four sections should be read in numerical order. The last four sections, covering analysis, simulation and layout, can be read in any order without sacrificing coherence.

## 3.0 Logic: The Standard Cell Library

The Logic library is a set of icons representing the most common Small Scale Integrated (SSI) and Medium Scale Integrated (MSI) circuits, including simple gates, adders, mutiplexers, register files, etc. Each icon is associated with a behavioral model (a Rosemary simulation procedure) and at least one layout generation procedure, either through a standard cell representation or standard generator.

Information about timing and size is provided for every cell and macro. All cells have a fixed height of 104$\mu$. The width is a multiple of 10$\mu$ which is called a track. Input loads are counted in standard loads (= 0.2pF) and output drives in standard drives (= drive of an inverter, i.e. a 2/20 n-device and a 2/50 p-device). All these numbers are subject to change. For more information about the logic library see: [DATools]<DATools7.0>CellLibraries24>LogicDoc.tioga.

### 3.1 Navigating in the Logic Library

Open the logic library on your color screen by doing the following:

1. Select the *Color Display* button which is left-most button in the static column of your screen (top right corner). A pull-down menu will appear. Choose the *8 bit CMos-B* entry.
2. In your datools subdirectory, execute the command,
   % cdread logic.dale

The logic library should appear on your color terminal. A control panel for the logic library with the banner, *Logic CMosB top level.* will appear in the right column of your black-and-white terminal.

The scale for logic is small enough that the items appear mostly as boxes. Here are five keyboard-mouse combinations that will allow you to see things and get around:

First, a word of warning to left-handed-mouse users: ChipNDale, the graphical editor for VLSI layout and schematics, is very finely tuned for right-handed-mousers. Consequently, these instructions specify the left hand for the keyboard and the right hand for the mouse. Try it that

way, some of the chording combinations are really difficult to reverse.

1. Centering selected objects (CtrlSpace)

   LeftClick on one of the white rectangular boxes. Note that the terminal viewer on the right-hand column of your screen has told you what you have selected. Now center that selection on your screen by pressing the CTRL key and the space bar at the same time (CtrlSpace).

2. Zooming out (TabSpace)

   Zoom out to the original magnification by pressing the tab and space keys at the same time (TabSpace).

3. Finding things in a large space: Search Object (DSpace, a case sensitive search)

   To list the contents of logic, call up the Directory Menu by pressing the *D* key and the space bar at the same time (DSpace). Choose the second option, *list directory*. The directory of logic will appear on your screen. Now select the *Search Object* entry. Look over to the terminal viewer. You are being prompted for a cell name. Find the nand3.icon in the Directory of Logic; shift-select it, then hit the carriage return (Return). The nand3.icon is selected and fills the screen.

   Note the use of mnemonics in selecting the letters for menus: D stands for Directory. Menus are generally invoked by pressing a letter and either the spacebar (Space) or the middleMouseButton (Middle).

   Most of the time, it is possible to get additional documentation explaining either:
   1) what the entries in a menu mean, or
   2) a keyboard equivalent for the menu item

   by moving the mouse above the above the titleBar of the pop-up menu and then moving it back into the menu item area. Try it on the Cell Menu (CSpace).

4. Changing Magnification <>

   To get a view of something besides the nand3.icon, find the < and > keys. The looks of these keys express their functions. The < key makes objects bigger. The > makes objects smaller. Hit the > key 5 times. You should now see most of the *released* section of logic.

   Icons in the *released* box are public. The name, size and shape of the icon, the position of the pins, and their general semantics are fairly stable. Any changes to these icons will be done with profuse apologies and a lot of advance notice. These are the icons to use for standard cell layouts.

5. Scrolling (SpaceMiddle)

   Center the word *Released* at the top of your screen by executing SpaceMiddle. SpaceMiddle must be executed in exactly the following order:

   1. Left hand on spacebar

   2. Right hand on middleMouseButton

   3. Move the mouse to draw a vector from your current position to the new position desired. In this example, this means you should draw vector from the word *Released* to just below the word CMosB in the terminal banner.

4. Take your right hand off the middleMouseButton.

5. Take your left hand off the spacebar.

The released section of the logic library should now be centered on your screen. To make the entire released section visible hit > one more time.

Note: Many users may find scrolling hard because the wrong timing of the left and right hands produces either stray wires or no action.

1. If Middle proceeds Space, Middle draws a wire and Space flips its orientation.

2. If Space proceeds Middle, but the left hand is taken off Space *before* the right hand comes off Middle, nothing happens.

The 5 keyboard combinations listed above should be enough to navigate in logic. At this point it is probably a good idea to practice a little. Move around. Push into and Pop out of a few of the other icons listed in the Directory.

Additional information about using ChipNDale can be found in the following documentation:

1. [DATools]<DATools7.0>CDDoc25>ChipNDaleDoc.tioga

This is the comprehensive reference guide for ChipNDale, the graphical editor for VLSI layout and schematics that you have been exercising.

2. [DATools]<DATools7.0>CDDoc25>ChipNDaleIntroduction.tioga

This is an introduction to ChipNDale to be used in the first hour of interactive usage.

3. [DATools]<DATools7.0>CDDoc25>CDCrib.tioga

This is a 4-page crib sheet that experienced users of ChipNDale find very handy.

All of this documentation is listed in [DATools]<DATools7.0>Top>CDDoc25.df. Get in the habit of opening the df file pertaining to the package of interest. It's the most convenient way to find all the files on a particular subject.

### 3.1.1 Printing

Nectarine creates Interpress masters from schematics and layout. Interpress masters can be printed on black-and-white and color print servers or stuffed into tioga documents. Execute the command, NectarineSchematics in your DATools subdirectory. A new *Nectarine* line will appear at the bottom of each ChipNDale Control Panel. To print the entire Logic Library on Sleepy, the color Versatic printer, click on top of the *Where* button until it says ColorVersatec. (Note: The *What* buttons flips through the options the same way. The *Copies* button can be edited.) Now click the *Nectarine* button. Nectarine creates an Interpress master and, if the selected printer cannot interpret the Interpress file format, Nectarine also creates a printer-dependent (.pd) version of the specified design. The default filename for these files is []<>Temp>Nectarine>fileName.fileType. After creating the required file or files, Nectarine sends it to the print server specified. Sleepy has a very short queue for print jobs. If Sleepy times out before it accepts file, you can try again later by executing the Chat command. For more complete information on printing see [DATools]<DATools7.0>Nectarine>NectarineDoc.tioga which is quite complete and doesn't need to be repeated here.

A second printing facility is available through the Hard Copy menu of ChipNDale. This menu is accessed by the pressing the *H* key and the MiddleMouseButton at the same time (HMiddle). After selecting type of file desired from the first HMiddle menu, a series of additional pop-up menus further refine the file specification. The HMiddle printing facility is preferable for very large chips. You might try both facilities to see which one you prefer.

### 3.2 Creating a New Design Using Standard Cells

Our first project will be to recreate the schematic for the oneBitAdder that is included in Logic using standard cells. Find the oneBitAdder.sch included in Logic using the *search object* entry from the DSpace menu.

Create a new viewer for a design by executing the command, cdNewCmosB your DATools Subdirectory. This should create a new, ChipNDale Viewer and a Control Panel for that ChipNDale Viewer. The Control Panel appears in the right-hand column of your black-and-white terminal. If the ChipNDale Viewer appears on your black-and-white terminal, move it to the color display by selecting the *color* button in its banner (and add a userProfile option which says: ChipNDale.FirstViewerOnColor: TRUE.)

There are two mechanisms for using objects from the Standard Cell Libraries to create new designs. *Include* actually copies the designated object from the Standard Cell Library into the new design. *Import* establishes a reference between an object in the new design and its referent in the cell library. In general, the second method is preferable because changes made to the referent are propagated to objects in the referencing design. In this example, we will import objects from Logic.

Before we can do the import, you need to know one more thing: ChipNDale distinguishes between design names and file names. A *Design* is the object created by the user. It may represent one or many circuits. A design is in a particular technology for example, CMos-B or NMos and consists of some geometry and a Directory. A *File* is is the storage unit for a design.

Now, back to our first project. To make things easier, first import the oneBitAdder to be copied by executing the following sequence of commands:

1. Left-click the mouse in the new, ChipNDale Viewer. In ChipNDale terminology, this is called placing the *input focus.*
2. Now hold down the X, Z, and middleMouseButton at the same time (X-Z-Middle). Look over to the Terminal Viewer it has printed out the command you have executed: "draw object of imported design." It is prompting for a "DESIGN name." Enter the name of the name design, Logic (the capital L is mandatory).

   Now look back to your ChipNDale Viewer. A pop-up menu is prompting for the name of the file that contains the Logic design. In this case the filename and design name are the same. Select the *Logic* entry that ChipNDale has provided.

   Look back to the Terminal Viewer it is now asking for the name of the object from Logic that you want to import. Type in its name, oneBitAdder.sch.

   Note: These interactions are case-sensitive so, when it is possible, it is safest to copy names out of the Directory of Logic.

To recreate the oneBitAdder, five gates must be imported: nand2.icon, nand3.icon, nor2.icon, nor3.icon, and nor4.icon. Since Logic has been established as the file from which cells are being imported, it is only necessary to do the following:

1. X-Z-Middle in your ChipNDale Viewer.
2. Select *Logic* as the Design for imports.
3. In response to the Terminal Viewer prompt, type the name of the object to be imported, or better yet, shift-select the name from the Directory of Logic.

   Note: Gates are imported to the position of the mouse when the X-Z-Middle command is executed. After importing the five gates, your *no name* viewer should look like this:

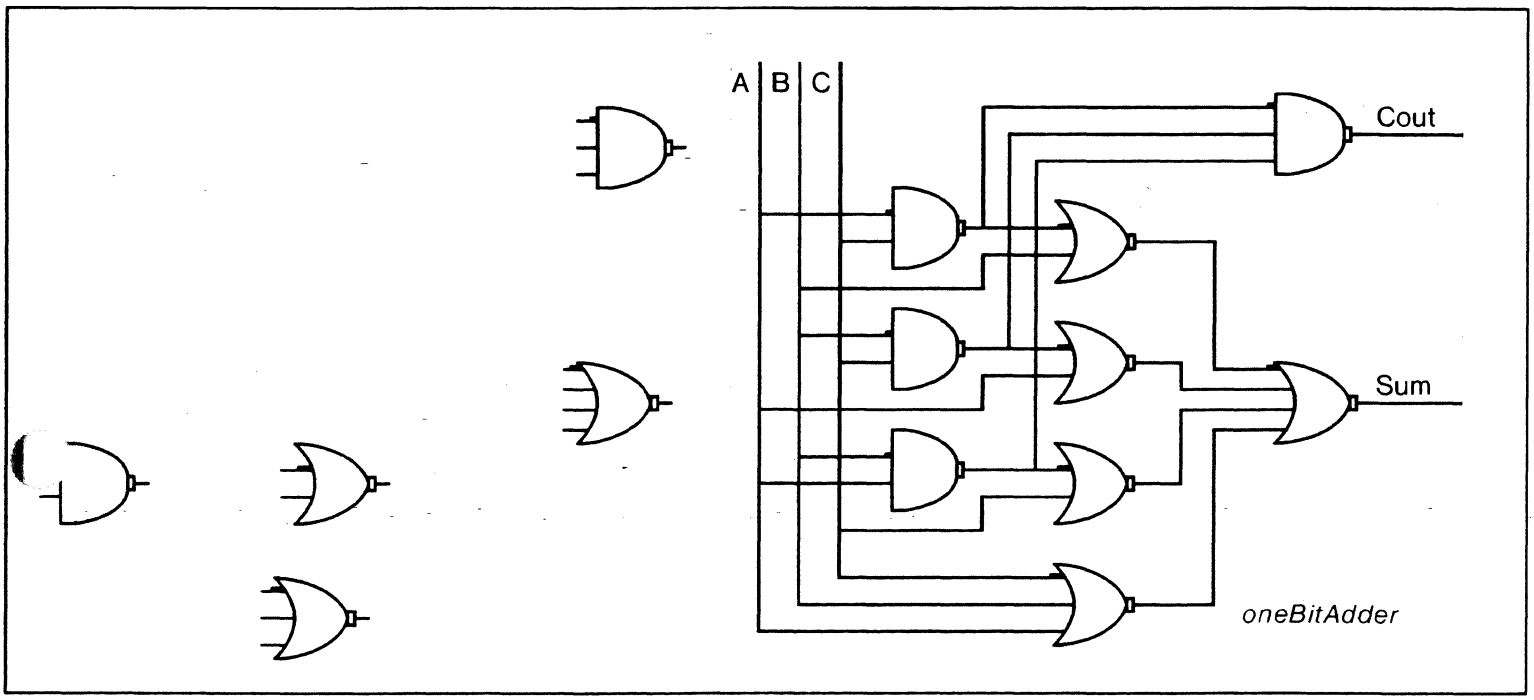



**Figure 1:** The screen after having imported the oneBitAdder and the five logic gates.

It's time to save your work. The IO menu is used for saving designs to a file. Invoke it with ISpace. Select the save option. This first time you will be prompted for a file name. Type in adderExample.dale then hit the *return* key (Return). Close the logic library to give your adderExample the whole screen.

Here are some additional commands to complete the oneBitAdder:

1. Get comfortable by selecting everything, centering and increasing the magnification

   To select everything hold down the space bar and the rightMouseButton at the same time (SpaceRight). Center your selection (CTRLSpace). Increase the magnification (<). Scroll your design to the right so that all the gates for the new oneBitAdder are visible (SpaceMiddle).

2. Moving individual objects

*By a Vector:*

Left hand on control. Holding the right hand down on the leftMouseButton select the object you want to move, then move the mouse in the desired direction. A vector will indicate the extent of the move. Take your right hand off the mouse when the vector looks right.

*By an Incremental step:*

The control key and the A, W, S, and Z keys are used together to move objects in incremental steps as follows:

ControlA = left
ControlW = up
ControlS = right
ControlZ = down

Look at the keyboard; this makes visual sense.

3. Copying

Make two more copies of the nand2.icon by doing the following:

Left hand on shift. Holding the right hand down on the leftMouseButton select the object you want to copy, then move the mouse to the point where the copy should appear. Take your right hand off the mouse. You should have another copy of nand2.icon. If it's not positioned quite right, use the incremental move command to scoot it around.

Now make another copy of nand2.icon and two copies of nor2.icon. Your screen should look like this:

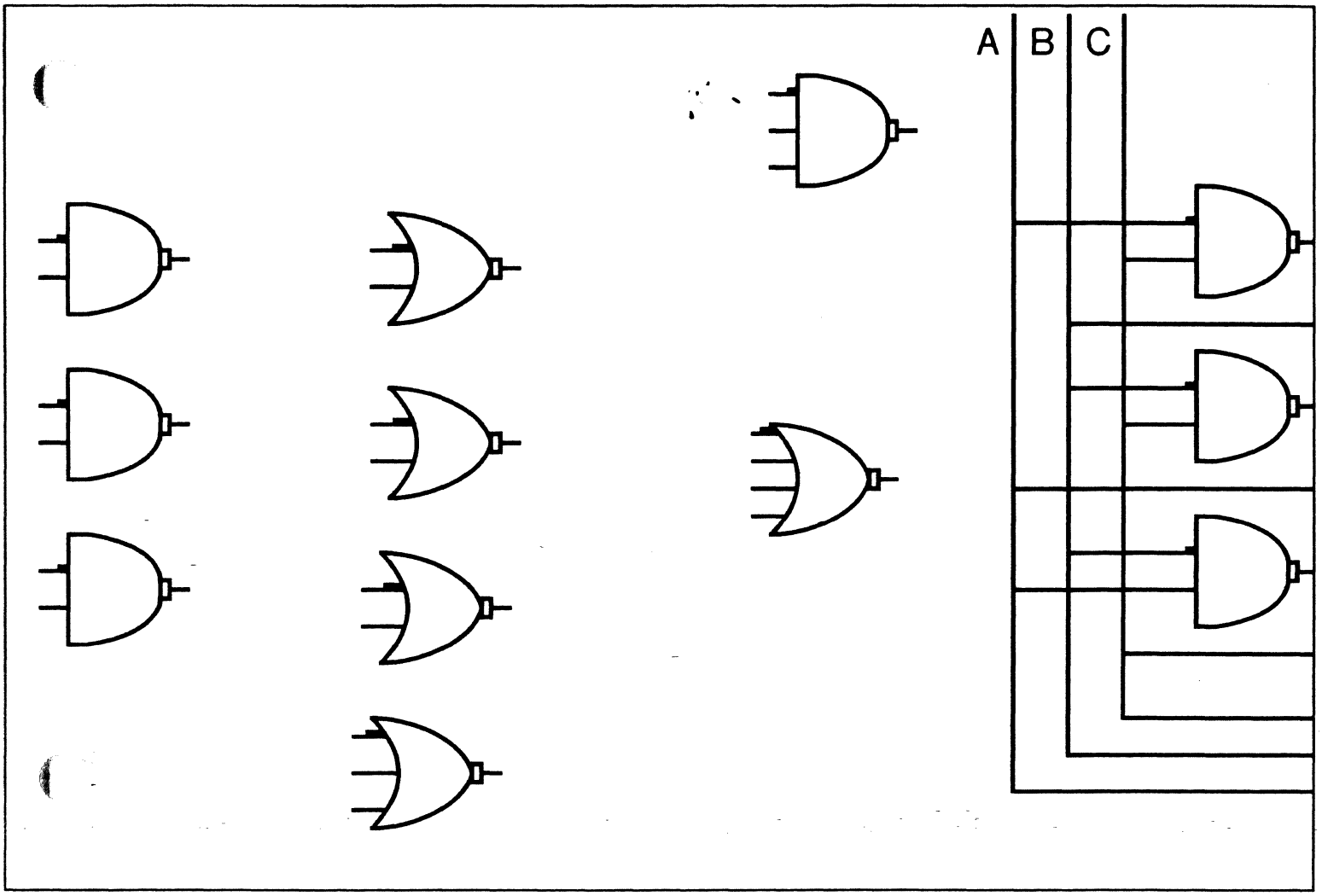



**Figure 2:** The color screen

Save again (ISpace). This is the last reminder about saving often.

4. Deleting (CTRL-D)

   To delete an object, first select it then (leftClick) and then hit the control key followed by the *d* key (CTRL-D).

   As objects are created they are added to the Directory of the design; however, when an object is deleted it is not removed from the design's Directory. To get unwanted objects out of the Directory, they must be deleted explicitly using the *prune complete dir* entry on the Directory (DSpace) menu. This process is not commutative. Objects must be deleted from the screen before they can be deleted from the Directory.

5. Un-Deleting (ESC-D)

   To restore an object that was deleted by mistake, hit the escape key followed by the *d* key (ESC-D).

6. Stopping a command while it is in progress (DEL)

Here's the scenario: while your left hand is on the spacebar you press the middleMouseButton instead of the leftMouseButton, thus drawing a wire instead of scrolling the screen. You can negate the action in progress by keeping your left hand on the space bar and hitting the DEL key. Try it.

7. Ticks (periodSpace)

Call up the ticks menu by pressing the *period* key and the spacebar (periodSpace). (Note that *periods* look like ticks.) Select the *four* entry from the menu. If you think it will be easier to draw wires with ticks on, leave the ticks on. Otherwise turn them off.

### 3.2.1 Drawing wires

For schematics, wires are drawn in black using a width of 4/8, that is 1/2 λ. (The reason that the Control Panel says *4/8*, not 1/2 λ, is to make it explicit the fact that λ can be subdivided into $8^{ths}$.) Look at the Control Panel for the adderExample. The top-left entry gives the current layer. If the current layer is not black, find the black button in the next row and middleClick on it. The current layer should now be black.

To draw a wire, hold down the middleMouseButton as you move the mouse. To draw a continuous wire that incorporates 90 degree angles, hit the spacebar as you move the mouse in a new direction. Try drawing some wires that bend. Try a square.

### 3.2.2 Connected vs crossing wires

When selected wires are shown at a very high level of magnification, they appear outlined in white, as Figure 3 illustrates. By selecting a wire segment and extending that selection to other wire segments, it is possible to determine exactly how a wire is composed.

Try the following introductory exercise:

Push into the oneBitAdder (push in picture, CSpace). Center the top left nand gate on your screen. Increase the magnification until the A, B, C inputs and top four gates of the oneBitAdder fill the screen. Select (leftClick) the vertical portion of the A wire. The vertical portion of A should now be outlined in white. Extend the selection to the horizontal portion of the A wire by clicking the rightMouseButton (rightClick). Both sections of the A wire are now outlined in white. Note that the outlined horizontal portion of the wire includes the vertical portion as Figure 3 illustrates. Using the spacebar to create 90° angles in wires results in this configuration.

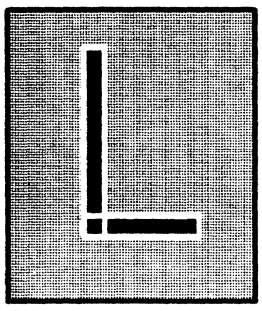


**Figure 3:** A continuous wire that was drawn using the spacebar to create a 90° angle. The white outline indicates that both sections of the wire are selected.

### 3.2.3 Connected vs crossing wires: The Rules

1. Two wires that are colinear and whose endpoints abut or overlap are connected. Figure 4a illustrates two wires whose endpoints abut. Figure 4b illustrates colinear wire segments that overlap. In both cases a connection exists.
2. Two wires that *almost* touch are not connected (Figure 4b).
3. Wires that cross without an explicit contact at their point of intersection are not connected (Figure 4c).
4. Crossing wires with a contact at their point of intersection are connected. There is a contact in the released portion Logic. Its name is ct.icon.

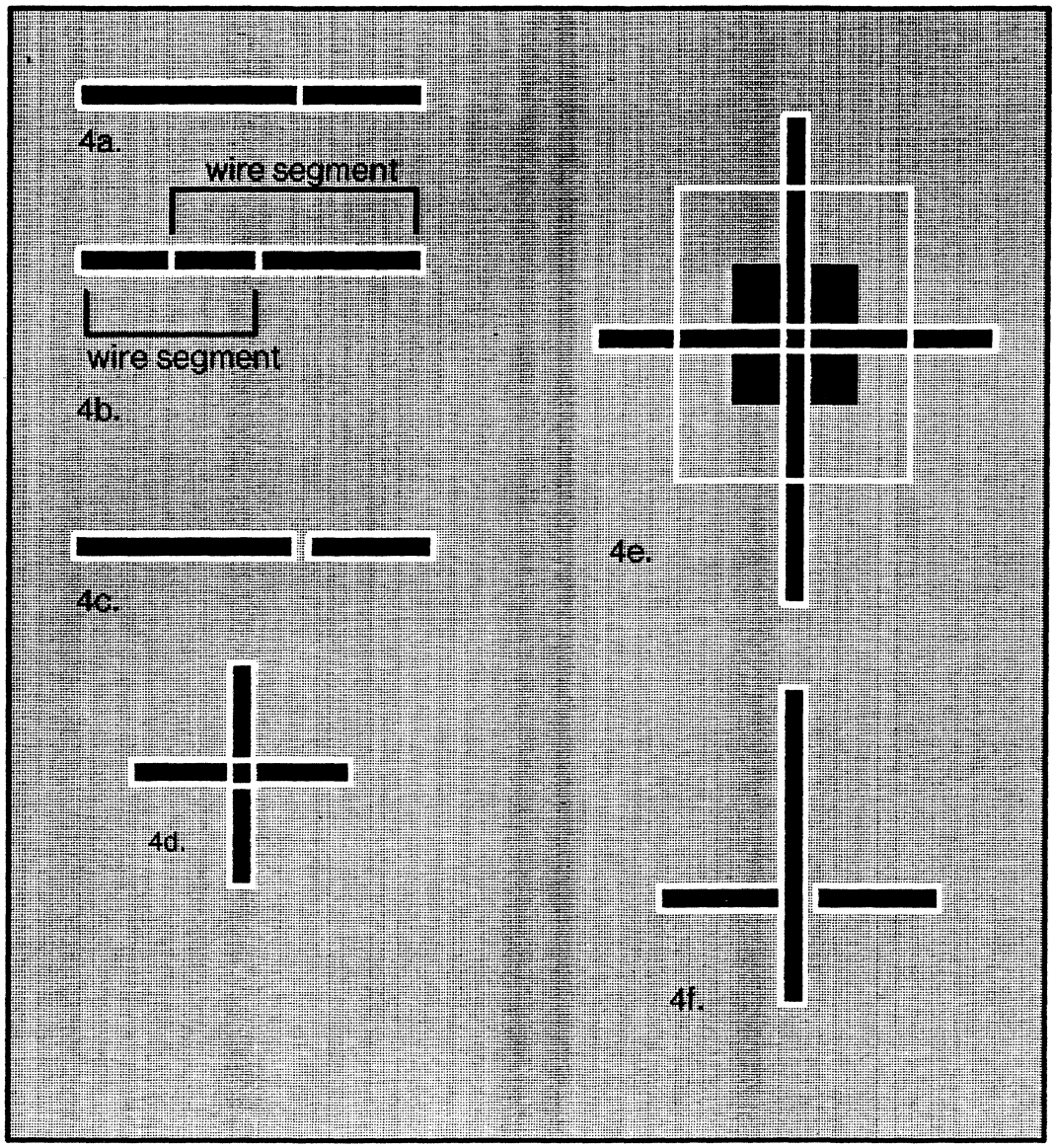


**Figure 4:** Crossing and connected wires. All wire segments are selected to illustrate the internal structure of the wires.

In Figure 4f the vertical wire and the left-hand horizontal wires are connected. The right-hand horizontal portion is not connected.

Figure 5 illustrates one difficulty that can arise when drawing wires. In 5a, the input wire to the inverter was drawn in two short segments instead of one long segment. This in itself presents no problem, because the two wire segments touch, they are considered connected. And unless the wires are selected, the designer will undoubtedly forget that A was drawn as two segements instead of one. But suppose at a later time, a second wire, B is drawn which is supposed to cross A without intersecting it. Unfortunately, B happens to cross A just where one of the two segments of A ends. A and B are now shorted together. Moral: always draw wires using long strokes. Check your drawings for unintended short wire segments.

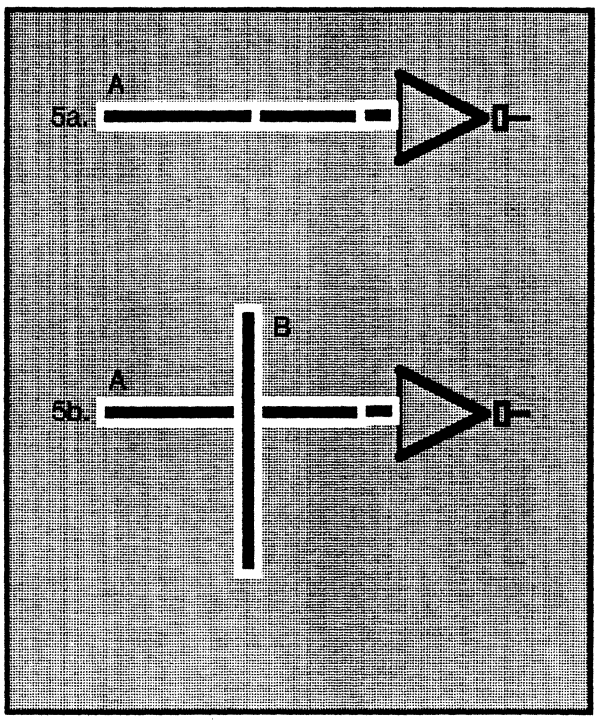



**Figure 5:** Unintentional shorting of two wires.

Always draw at a high level of magnification. In this example, an appropriate level of magnification will force the oneBitAdder you are copying out of view. There is a copy of it in Figure 1 above. To find it, first split this viewer then search the word, *Figure 1.* The oneBitAdder should be visible.

To draw the wires in the oneBitAdder, first draw the long "L" portion of the A wire from the top left corner to the bottom nor3.icon, using the shift key to create the 90° angle. Then draw the horizontal portions of A. Continue in this fashion for the B, C and internal wires. Extend the length of the output wires from the nand3 and nor4 icons.

### 3.2.4 Attaching names: Satellites and Expressions

Satellites provide a way to associate arbitrary textual information including names and expressions with ChipNDale entities. There are two kinds of satellites:

A. Instance Satellites: An instance satellite is a Chipndale text instance that has been associated explicitly with another graphical Chipndale instance called the satellite's *master*; a master along with its satellites is called an *instance group.*

B. Object Satellites: An object satellite is a text instance that has been associated with the containing cell or design.

Both instance and object satellites can be made comments in the programming sense of that word. Comment satellites are used when a piece information should be included in a design, but this information should not be interpreted by programs that analyze the design. Comments remember their masters.

Expressions also come in the instance and object varieties. The only difference between Satellites and Expressions is that Satellites are visible and Expressions are not. Expressions are provided for those occasions where placing satellites would clutter up the entity on which they are being placed. Expressions are generally used on icons. To illustrate this point, push into one of the nor or nand icons and select its top input wire. Call up the satellites menu (LSpace). Select the *Show Instance Expressions* entry. The Terminal Viewer should say, "name ← "I-A"."

Because the names associated with the input and output wires in the oneBitAdder should be visible, they will be stored as satellites. Select the wire that is called *A* in the oneBitAdder. Call up the satellites menu (LSpace). Select the *draw instance satellite* entry. The Terminal Viewer will prompt for a text string. Type in the appropriate text string (CR). The text string should appear *about* where you want it. Select it, then scoot it around using the Incremental Move command. Name the three input and two output wires for the oneBitAdder.

In the terminology of Core, the five wires that have just been named are *public* wires. These are the wires that connect to other cells. They are at the *interface* of the oneBitAdder. Wires that are not at the interface of a cell are called *private.*

### 3.2.5 Creating Cells

Cells associate entities and give that association a name. A cell is created by drawing its bounding box and using the Cell-Menu to create and label the association. To make the oneBitAdder a cell, it must be selected. This selection is done with an *Area Select.*

1. Area Select

Area Select is accomplished by holding down the leftMouseButton and moving the mouse to sweep out the rectangular area to be included in the cell. The area selected does not have to be a precise bounding box for the cell.

Now call up the cell menu (CSpace). Select the *create cell* entry. The Terminal Viewer is

prompting for a name. Enter the name, "oneBitAdder" (CR).

The cell just defined is a ChipNDale object. Because an object has been defined, it is possible to attach satellites to it. While pushed into the oneBitAdder, call up the LSpace menu and select the *draw object satellite* entry. Give the cell a name by typing in the name, oneBitAdder, in response to the Terminal Viewer's prompt. Make that name a comment (CTRL-\.) Pop out. ChipNDale provides several options when popping out of a cell: To store the changes made, select the *replace* entry in the pop-up menu that appears. The *flush* entry throws out changes. The *new cell* entry is used when the current cell is a modification of an earlier cell that should remain unchanged; if the *replace* option had been selected, changes made to the new cell would be propagated to all other copies of that cell.

## 4.0 Extraction

Sisyph is a schematics extractor that produces a structural description of a circuit from schematics drawn in ChipNDale. Sisyph processes the geometry contained in ChipNDale and uses this geometrical information to define the connectivity of the circuit. The circuit description that Sisyph creates is called Core. Core exists in main memory. This Core structural description does not actually include geometrical from ChipNDale; instead, it includes pointers to that geometrical information in its property list.

### 4.1 Sisyph: Extract

To extract the oneBitAdder call up the OSpace menu and select the *Sisyph: Extract* entry. The results of the extraction will be printed in the Terminal Viewer.

#### 4.1.1 Sisyph Some Important Terms from the Extraction

The Sisyph extraction has created a oneBitAdder *CellType* from the oneBitAdder schematic. The *Public wire sequence* of the oneBitAdder CellType contains the seven elements: A, B, C, Cout, Sum, Vdd and Gnd. The *Internal wire sequence* of the oneBitAdder includes all of its public and private wires; it contains fourteen wires. The nine logic gates that make up the oneBitAdder are *CellInstances* in the Core data structure. These CellInstances are contained in a *RecordCellType* which is a sequence of CellInstances. Each of the nine CellInstances has an *Actual wire sequence*. The Actual wire sequence of the individual gates points into the Internal wire sequence of the oneBitAdder. For example, looking at the nand3 CellInstance, the I-A input corresponds to [12] in the internal wire sequence. The output, X, corresponds to Cout in the internal wire sequence. It is by enumerating the actual wire sequence of each CellInstance that all the connections in the oneBitAdder become known. For more information about the Core data structure, see, [DATools]<DATools7.0>Core>CoreDescription.tioga.

### 4.2 Sisyph: Make Icons

An icon is a schematic that represents another schematic. Its purpose is to simplify the pictorial representation of the schematic being represented. The looks of many icons have meaning for designers. For example, a triangle with a bubble on its output is universally recognized as an icon for an inverter. The same inverter might be represented by a stick diagram of a pull-up and a pull-down transistors. As circuit diagrams become more complex, icons are used to represent more

complex circuit components, thus maintaining readability.

Icons can be associated with schematics that have actually been drawn as well as fictitious schematics whose Core structural descriptions are computed directly by the code attached to the icon. This attachment of code to icons is part of a general mechanism that permits arbitrary CEDAR expressions to be attached to geometric objects to control their extraction.

To create a fourBitAdder from the oneBitAdder, the oneBitAdder should first be represented as an icon.

Call up the OSpace menu, and select the *Sisyph: make icons*, entry. In the second pop-up menu that appears select the first entry, *Create icons from schematic*. Sisyph will extract the schematic and return a vanilla icon. The looks of this icon can be altered. Move the carry-in to the left side of the rectangle, the carry-out to the right, and the sum to the bottom. To demonstrate that the oneBitAdder icon has been associated with the correct schematic, first select it then call up the Cell (CSpace) Menu. Select the *push in schematic* entry. You should be pushed into the oneBitAdder schematic.

1. Repetition

A repetition places a set of objects at regular intervals along a drawn vector a specified number of times. Groups of objects and repetitions themselves can be used as the basis for repetitions. The repetition feature is used to create uniform structures quickly and easily. Experienced designers generally use the repetition feature to draw 5 or more copies of an object and choose the copy command for fewer copies. The repetitions will be used here to illustrate its utility.

To make 3 additional copies of the oneBitAdder icon do the following:

1. Select the oneBitAdder icon.

2. Hold down the "=" key and draw a vector from the top left-corner of the adder icon to a point comfortably beyond the top-right corner, as Figure 6 illustrates.

3. Answer the Terminal Viewer prompt for the number of repetitions you want, "4" (CR). ChipNDale creates a new cell which contains the four instances of the oneBitAdder icon. Logically, these four oneBitAdders are at the same descriptive level. The gray rectangle enclosing the four instances in Figure 6b indicates this. Any additions would be at the next higher level of the cell hierarchy: however, the connections between the carry bits of the individual should be at the same level of the hierarchy as the adder icons. To flatten the hierarchy, push into the cell and select the *expand* option. This flattens the cell, leaving four instances of the original oneBitAdder icon. (See Figure 6c.)

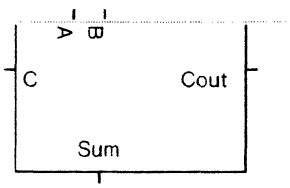



**Figure 6a:** Draw a vector that specifies the spacing between objects of a repetition.

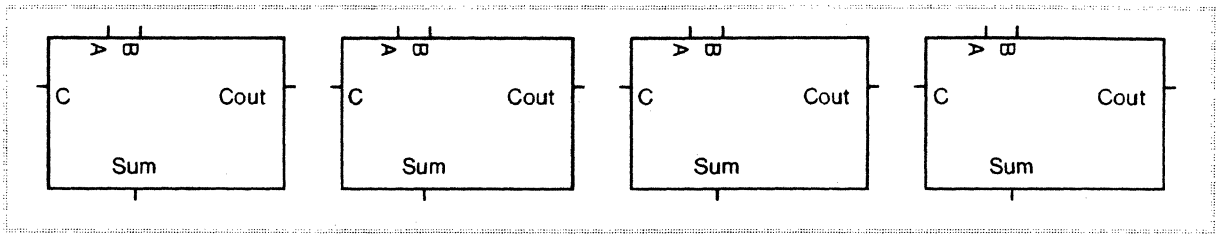



**Figure 6 b:** A new fourBitAdder cell. All four of the adder icons are contained in one rectangle.

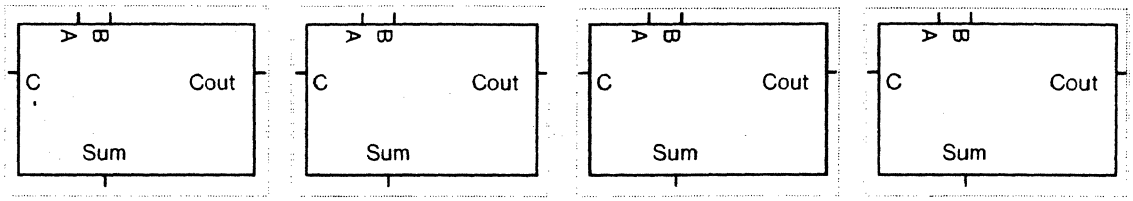



**Figure 6c:** The expanded fourBitAdder -- four copies of the oneBitAdder with no additional hierarchical semantics.

Connect the carry bits. Name the low-order carry by selecting it. calling the Satellites Menu (LSpace), and choosing the *draw instance satellites* entry (I-Middle) to attach the name CIN.

To draw the A inputs as a 4-bit bus, first draw the long horizontal portion of the wire, then call up the Programs on Rectangles menu with PMiddleMouseButton (PMiddle). Select the *PatchWork generator* entry. In the next menu select the *Parameterized Bus*. The Terminal Viewer is prompting for the size of the parameterized bus. Answer 4 (CR). To extract the low order wire from the bus, call up the PMiddle menu again and select the *PatchWork generator* entry. Then select the *Parameterized Extractor* entry. Follow the Cedar and Dragon conventions for numbering bits by answering the Terminal Viewer prompt, "Parametrized index of the extracted wire?" with the number 3. Answer the next Terminal Viewer prompt, "Size of the parametrized bus?", with the number 4. Place the 4/3 icon and connect it to the A input of the first icon. In drawing the connection make sure the connecting wire does not short to the 4-bit bus. Figure 7a illustrates a proper connection. Figure 7b illustrates a wire that is shorted to the bus.

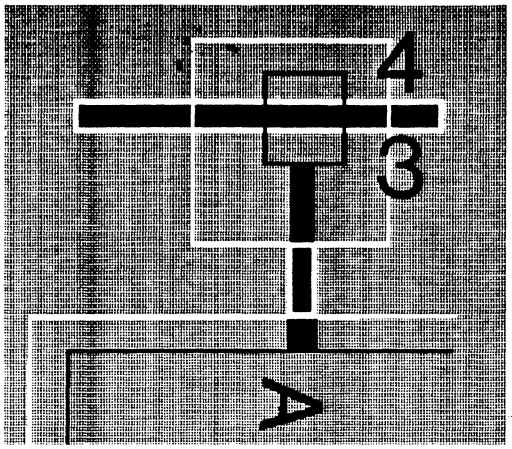



**Figure 7a:** Input A is correctly extracted from the bus.

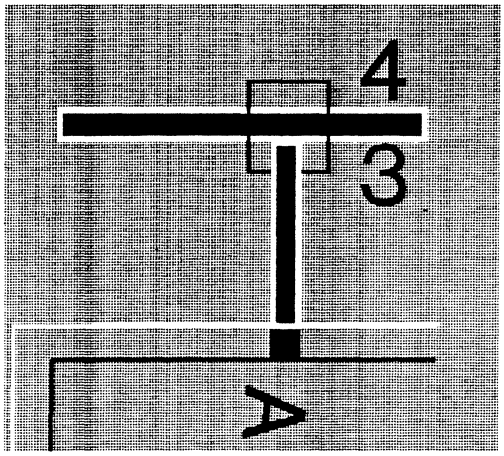



**Figure 7b:** Input A is shorted to the bus.

Icons produced by the PatchWork generator can be copied but NEVER edit them. They are full of magic and guaranteed *not* to work if they are not produced by the PatchWork generator. Finish drawing the A bus using the Patchwork generator to create icons for the 4/2, 4/1 and 4/0 extractions. The icons created for the A bus can be copied for B. Name both busses by attaching instance satellites.

2. Mirroring - The Transformation Menu (TSpace)

To create the extractions for the Sum, it is necessary to mirror the icons used for A and B. Look at Figures 8a and 8b to convince yourself of this. Name the 4-bit sum, SUM.

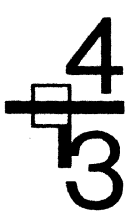



Figure 8a


Figure 8b

**Figures 8a and 8b:** Figure 8a should be used for the A and B inputs. Figure 7b should be used for the Sum.

The Transformation Menu can be used to mirror and rotate the extraction icons. To mirror each icon, first select it, then call up the Transformation Menu (TSpace). Select the *mirror y* entry. Now finish drawing the sum line. (Patchwork is smart enough to figure out that the Sum wire is a bus of size 4 without explicitly adding this information.)

Name the public wires by attaching instance satellites to them. To make the fourBitAdder into a cell, draw its bounding box and select the *create cell* entry from the Cell Menu (CSpace). Name the cell fourBitAdder. To add two object satellites to the fourBitAdder, push into it then call up the Satellites Menu (LSpace). Select the draw object satellite entry (O-Middle) and enter the cell name as a satellite. Make it a comment, (Control\). Pop out, selecting the *replace* option. Your fourBitAdder should look like this.

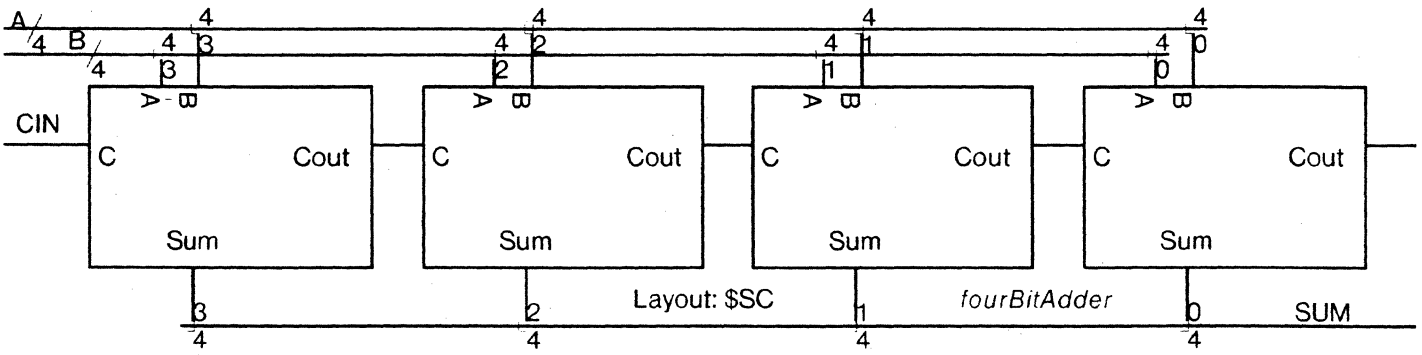



**Figure 9:** The completed fourBitAdder.

To create a fourBitAdder CellType, call up the OSpace menu and select the *Sisyph: extract* entry. The fourBitAdder CellType contains four CellInstances of the oneBitAdder CellType. Note that the public wire contains 5 elements: Vdd, Gnd, A, B, and Sum. A, B, and Sum are, in turn, composed of 4 elements. Each CellInstance has an actual wire sequence that points into the internal wire sequence of the fourBitAdder CellType. The actual wire of each oneBitAdder CellInstance specifies to which of the four elements of the structured wires it is connected.

## 5.0 Sisyph Extract and Static

Static is a static electrical analyzer that detects logical errors in a schematic. The errors are printed in the Terminal Viewer. For example, Static would flag a node that can never be set to 0 or 1, a degraded input that is used to control a pass transistor, a node with only one electrical connection. See [DATools]<DATools7.0>Static>StaticDoc.tioga for a little more information on what Static does.

To demonstrate that Static works, delete a bit of the Cout wire so that it no longer touches the edge of the oneBitAdder's bounding box. Now select the *Sisyph Extract and Static* entry from the OSpace menu. Note that the Terminal Viewer prints the message, "Cout in cell oneBitAdder has only one connection."

## 6.0 Sisyph and Rosemary

Rosemary simulates the behavior of integrated circuits. It performs a cycle by cycle simulation of a circuit and determines the value, either 0 or 1, for each input and output at each clock cycle. Rosemary performs no analysis of timing. For example, it does not consider the propagation time of the basic logic gates. Its focus is circuit logic. Timing issues are covered by two other simulators, Thyme and Spice. A third tool, Mint, performs both simulation and analysis.

The user must provide Rosemary with a test procedure. For simple cells, the test procedure takes the form of clocks specifying wave forms attached to each input wire. For complex circuits, the test procedure is a Cedar Program. For circuits where a reasonably small number of test vectors are needed to verify correct functioning, a file containing test vectors can be supplied to Rosemary by using the Oracle.

In addition to a test procedure which the user must provide, the user may provide models for abstract blocks that are connected to the CellType that is being simulated: A state initialization procedure may be written to allocate state storage of abstract blocks. And, evaluation procedures may be written to model behavior.

### 7.0 Rosemary: Simulating the OneBitAdder Using RoseClocks

The oneBitAdder will be used to illustrate the creation of a test procedure from roseClocks. (Refer to Figure 10 to see what things should look like.) Do the following:

1. Copy the oneBitAdder.
2. Import roseClock.icon from Logic.
3. Connect a roseClock to each input wire.
4. Change the wave form of the A input by:
    a. selecting it

stop the test. Note that the values for Sum and Cout are being displayed. To see the wave forms for Sum and Cout open the oscilloscope icon. *Proceed* will continue the test. *Abort* ends it. Now try the *Single Eval* button in the adder.sim viewer.

The typescript viewer was originally designed for communication with Rosemary; however, the two most heavily used typescript commands, the *value* and *add* commands, have been superceded by interactive equivalents. Here are their interactive versions:

1. Value (V-leftClick)

   *Value* prints the current value of the selected wire in the Terminal Viewer. To determine the value of a given wire, hold down the *v* key then select the wire whose value you want to know. If the *v* key is kept depressed, it is possible to continue selecting wires and have their values printed in the Terminal Viewer.

2. Add (OSpace, Schematic Simulation Add To Plot)

   *Add* adds the selected wire to the to the wires displayed in the plot viewer. First select the wire you want to add, then select the *Schematic Simulation Add To Plot* entry from the OSpace menu.

To learn about the other typescript commands see:
[DATools]<DATools7.0>Rosemary>RosemaryDoc.tioga.

### 6.2 Rosemary: Using the Oracle To Provide Test Vectors

For circuits where a reasonably small number of test vectors are needed to verify correct functioning, the Oracle may be used to provide test vectors to Rosemary. The Oracle has three required parameters, an input bus (in), an output bus (out), and the name of the file containing test vectors (id). The Oracle's default behavior is to stop the first time it finds an error; however, if a fourth optional parameter (log) is used, the Oracle records errors and continues.

The fourBitAdder will be to illustrate the use of the Oracle. Do the following:

1. Create an icon to represent the fourBitAdder (Sisyph: make icons). Name it 4BitAdder.
2. Import oracle.icon and roseClock.icon from Logic.
3. Connect the inputs and outputs of 4BitAdder to the Oracle's input and output busses. To do this, think of things from the Oracle's point of view: the circuit's inputs (A, B and CIN) are the Oracle's outputs; that is the Oracle drives its outputs and compares them to its inputs (Cout and SUM). Because the Oracle accepts only 1 input and 1 output bus, the multiple inputs and outputs of any circuit must be combined into a single input and a single output bus by using composers.
4. Using Figure 11 as a model, get a composer of size 3 by calling the PatchWork Generator (PMiddleMouseButton). Connect A, B, and CIN to the left side of the Oracle (the output side). Get a composer of size 2 and connect Cout and SUM to connect the right side of the Oracle (the input side). Connect a roseClock.icon to the bottom of the Oracle.

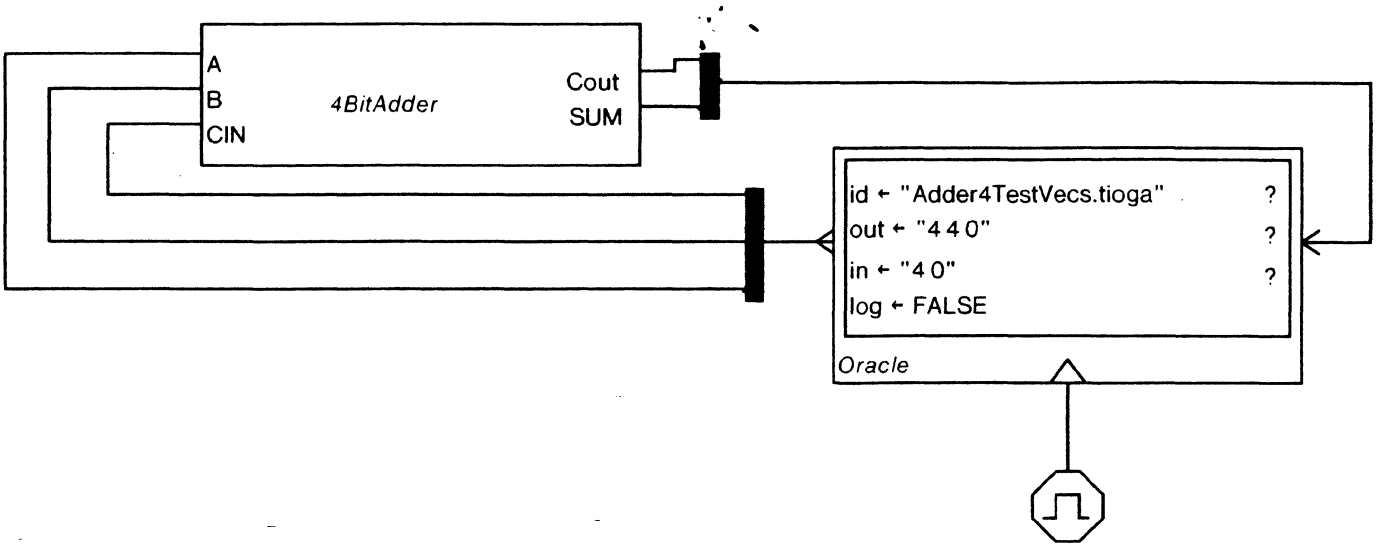



**Figure 11:** Using the Oracle to provide test vectors to Rosemary.

5. Now create a file of test vectors. Test vectors are written in hex. There's a column for each input and output. Inputs and ouputs are separated by a vertical bar. A period at the end of the test file will cause the test vectors to be run 1 time. Without a period, the simulation will cycle through the test vectors repeatedly. Here are 2 sample entries for a test file:

   *A B CIN/SUM Cout*

   8 8 0 | 0 1

   2 3 0 | 5 0

6. Add an instance satellite to the Oracle that assigns the name of the file containing test vectors to *id*. Add 2 more instance satellites for *in* and *out*. The thicker stub that distinguishes the bottom connection on the composer indicates that it is the $0^{th}$ wire of a wire sequence. *Out* is described by the string, "4 4 0" because A and B are 4-bit busses and CIN is an atomic wire. The statement, "in ← "4 0"" means that Sum is a 4-bit bus and Cout is an atomic wire. If you want to understand why atomic wires have a size of 0 (not 1), see Section 3.0 of [DATools]<DATools7.0>Core>CoreDescription.tioga.

7. Make this schematic into a cell. Call it 4BitAdderTest.sch. Select *Sisyph Extract and Rosemary* from the OSpace menu to get the Rosemary test viewers. Start the test. If the test completes successfully, a debugger will appear with the RosemaryImpl.Stop signal showing the message, "Oracle completed successfully." If there is a disagreement between the test vectors and the Rosemary simulation, a debugger will state the disagreement and the test vector file will be opened with the selection positioned at the offending vector.

### 6.3 Rosemary: Naming Conventions for Wires

The precise syntax for wire names is quite complex. It is described in the *Theory* section of [DATools]<DATools7.0>Core>CoreFlat.mesa. This section gives a general overview of the naming syntax.

Here is the name of an atomic wire in the oneBitAdder that is referenced by the 4BitAdderTest.sch.

/2(fourBitAdderLayout)/0(fourBitAdder)/0(oneBitAdder)*1.[11].

The use of slashes in this name is analogous to the use of slashes in filenames, with each additional slash describing the next lower level in the CellType hierarchy. Each level of the CellType hierarchy may be described by either the name of the CellType or a number indicating its instance number, or both. For example, typing a *c* in the Rosemary Script viewer describes the default (top level) CellType which is the 4BitAdderTest.sch. The 4BitAdderTest has 4 CellInstances:

```
CellInstance : ClockGen
    Actual wire:  Clock: [3](2428550↑); RosemaryLogicTime:
RosemaryLogicTime;
CellInstance : Oracle
    Actual wire: CK: [3](2428550↑); In: In; Out: Out:
CellInstance : fourBitAdderLayout
    Actual wire:  Vdd: Vdd; CIN: Out.CIN; SUM: In.SUM; B: Out.B; A:
Out.A; Cout: In.Cout; Gnd: Gnd;
```

And, typing either "c /1" or "c /Oracle" or "c /1(Oracle)"will produce the core structural description of the Oracle which is one of the CellInstances that makes up the 4BitAdderTest.sch.

A wire path is terminated by a period, and the *number* or name of the wire is given. For example:

/2(fourBitAdderLayout)/0(fourBitAdder)/0(oneBitAdder)*1.Sum.

Although the description of the syntax in CoreFlat.mesa says that wire numbers must appear in brackets, this is incorrect; wire numbers may be unbracketed.

## 7.0 Sisyph Extract and Thyme

Thyme, SPICE (Simulation Program Integrated Circuit Emphasis) and Mint are tools for verifying the timing of integrated circuits. Thyme, written in CSL, and SPICE, developed at UC Berkeley, are quite similar. Both use a highly detailed electrical model; as a result, they can only handle a small number of transistors in a reasonable amount of time. For example, SPICE handles 4 to 10 transistors well, and becomes impossible with over 100 transistors (an hour to simulate for 1 nanosecond).

In contrast to Thyme and SPICE, Mint has a simplified electrical model that includes only transistors and capacitors. It was developed to provide fast verification of logic and timing features and can handle a fairly large number of transistors (10,000) in a reasonable amount of time. If a chip is constructed entirely of standard cells, Mint simulations should be adequate. (It is safe to assume that the standard cells have been simulated by Thyme.)

### 7.1 Thyme: Simulating the OneBitAdder Using Icons from the Electrical Core Classes

Thyme and Mint simulations are performed by drawing a schematic and then adding a Control Panel and icons to represent circuit components, probes, and signal generators. The original schematic, the added electrical icons, and control panel are encapsulated in a new cell for the simulation.

To simulate the oneBitAdder using Thyme, do the following:

1. Make a copy of the oneBitAdder.

2. Using Figure 12 as a model, extend the lengths of the public wires, A, B, C, Cout and Sum.
3. Read in the electrical icons by executing, cdread ee.dale in the appropriate Command Tool. (For a complete explanation of the electrical icons, see, [DATools]<DATools7.0>ElectricalCoreClasses>ElectricalCoreClassesDoc.tioga.)
4. Designate ee.dale as the design for imports: X-Z-Left.
5. Import the Voltage, Pulse, RectWave and VProbe icons.

   A. The Voltage signal generator provides a source of DC voltage. Its units are Volts, and its default value is 5.0. Demonstrate this to yourself by selecting it and listing its object expressions (LO-Left). Attach voltage.icon to the A wire.

   B. The Pulse signal generator creates a signal pulse on the wire to which it is attached. Attach pulse.icon to the B wire. The pulse's object expressions include variables for onLevel, offLevel, period, width, tRise, tFall, tDelay. Change tDelay from the default of 10 nanoseconds to 15 nanoseconds by adding an instance satellite to the selected pulse.icon (I-Middle). Change the font size of this expression by selecting the *font* entry in the control panel and selecting *Helvetica8 2/8*. Then, with *tDelay ← 15* selected, type CTRL-F. (*F* stands for font.)

   C. The RectWave signal generator provides a periodic rectangular wave. Attach it to the C wire. Note that RectWave signal generator has the same variables to control its shape as the Pulse signal generator. Change tRise to 8 nanoseconds and tFall to 3. Now change the default font back to Helvetica8 4/8.

   D. The VProbe is a one terminal device whose active terminal is attached to the node whose voltage is to be plotted. To have the simulation plot all public wires, attach a VProbe to each of them.
6. Now import the Control Panel and position it below the schematic. Note that the panel has three columns. The first column states the parameters for the simulation. The second gives default values for the simulation. And, the third provides a space to override the defaults. You may want to override tMax ← 100.0 because a 100 ns simulation of this adder will take about 45 minutes. Do this by attaching an instance satellite to the panel. Add a simulation title. (The tStep value is for SPICE simulations.)
7. Names propagate up the CellType hierarchy iff there is no conflict. In the example being created, the VProbe and Voltage icons attached to the A wire create a naming conflict, so to have the public wires named for this simulation, each must be renamed at the higher level. Do this.
8. Create a new cell that includes the electrical icons. Call it adderTest.
9. Run the thyme simulation by executing the following commands in the appropriate Command Tool:

   A. install Thyme

   B. run ElectricalCoreClassesImpl

   C. LoadStdCellsCmosB
10. Select the *Sisyph Extract and Thyme* entry from the OSpace menu. Two new viewers are created:

The first, a Thyme viewer, appears in iconic form. The use of this viewer is explained in [DATools]<DATools7.0>NewThyme>ThymeDoc.tioga: however, that documentation is out-of-date. The important things to know are:

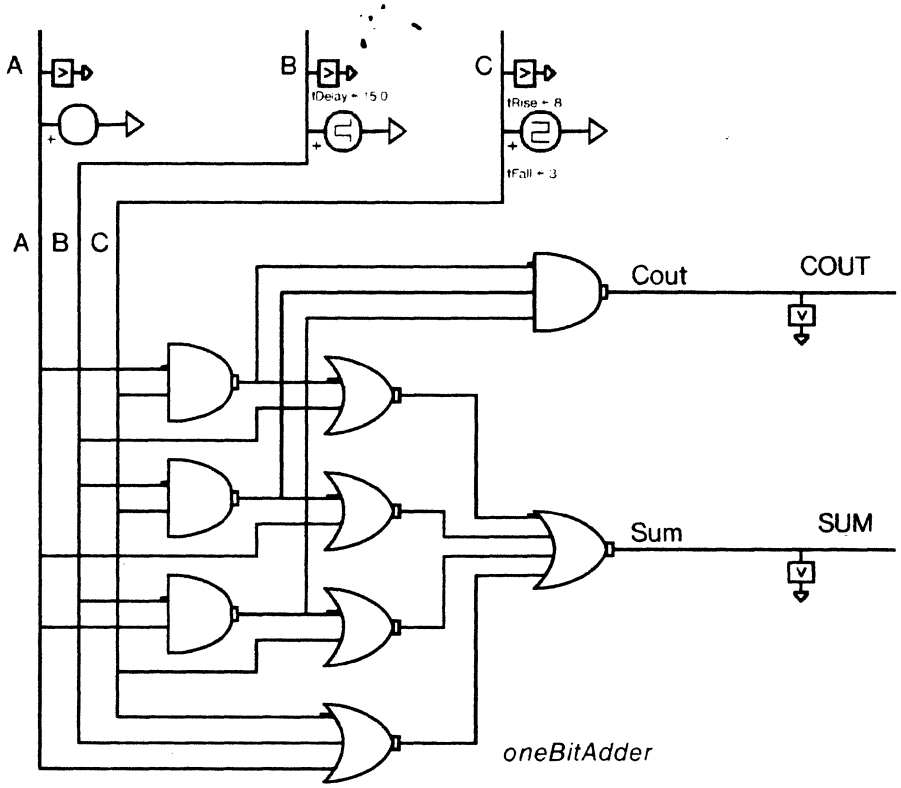
   A. The *Stop* button aborts the simulation without saving anything.

B. The *Dump* button aborts the simulation but also saves the calculation that has been done so far, to be used as the initial condition of future simulations.

C. The *details* switch supplies time/step information. It's default value is on. It can be turned off by clicking over it. (This changes the background of the button to white.)

D. The *EchoInput* switch echos the input statements that Thyme is reading. Its default value is off. You can turn it on by clicking on it. (ThymeDoc is about this!)

The second viewer creates a graph of the designated wires. A lot can be done with the graph. For example, it's possible to get a file of the values used in constructing the graph by: typing a filename, selecting the filename, and then selecting the *list* button. The file will open on your black-and-white terminal. For additional information see, [DATools]<DATools7.0>Graph>GraphDoc.tioga.




| Simulation Parameters | | |
|---|---|---|
| Parameter | Defaults | Current Values |
| Simulation start time | tMin ← 0.0 | |
| Simulation stop time | tMax ← 100.0 | tMax ← 50.0 |
| Simulation step time | tStep ← 0.5 | |
| Graph Title | title ← "" | Input for timing analysis using either Thyme or Mint. |
| Vertical axis min (volts) | yMin ← -1.0 | |
| Vertical axis max (volts) | yMax ← 6.0 | |
| Horizontal axis time scale | tScale ← 1.0 | |
| Power supply current scale | iScale ← 1.0 | |

**Figure 12:** Simulating the oneBitAdder using electrical icons from [DATools]<DATools7.0>ElectricalCoreClasses>EE.dale.

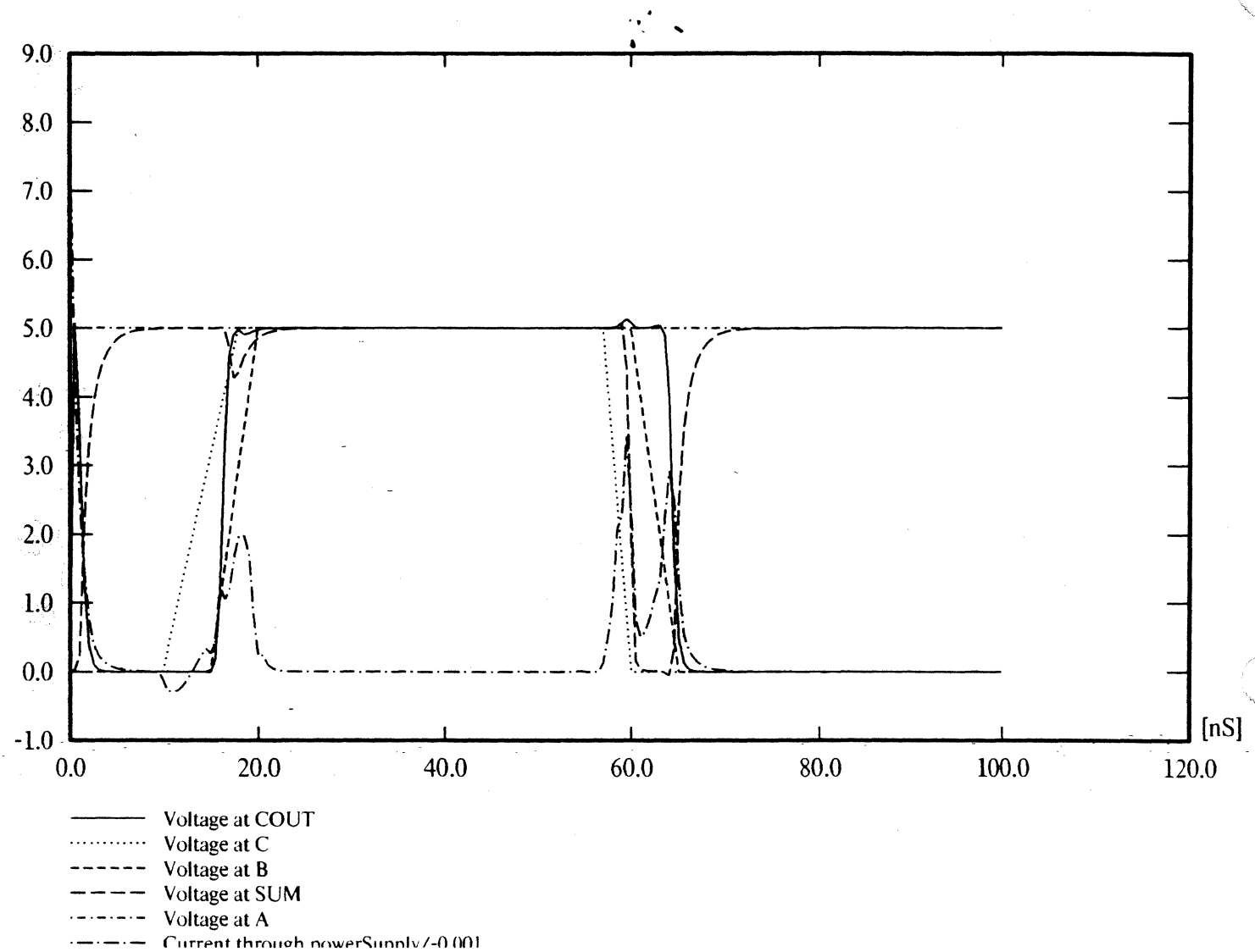



**Figure 13:** Thyme plot of the public wires for the oneBitAdder.

## 8.0 Sisyph Extract and Layout

Layout procedures are stored as values in a table composed of key/value pairs. The key is a layout atom. To create layout for the fourBitAdder that uses standard cells exclusively, the layout atom, $SC (Standard Cell) must be specified as an object satellite of the design. If the layout for a schematic is composed entirely of standard cells, the layout program generates layout by examining the layout atom stored at the highest level in the CellType hierarchy. To be sure that the layout atom is at the top level, many designers add another level to the CellType hierarchy and put only the layout atom in this highest level. Try this doing the following:

1. Select the fourBitAdder.

2. Select the *create cell* entry from the Cell Menu. Name the new cell fourBitAdderLayout.
3. Call up the Satellites Menu (LSpace) and select the *draw object satellites* entry (O-Middle). Answer the terminal prompt: Layout: $SC.
4. Pop out, selecting the *replace* option.

To generate the layout do the following: **Check if *Install SC* works**

1. Install the Standard Cell Library by executing the command, *Install SC* in your datools subdirectory.
2. Call up the OSpace menu and select the *Sisyph Extract and Layout* option.

The Terminal Viewer displays messages that indicating the progress of layout generation. The effective standard cell area including routing will be roughly 2.5 to 3 times the area of the individual standard cells in the design. The completed layout will appear in a new ChipNDale Viewer. To find out more about generation of layout see, [DATools]<DATools7.0>Top>PWCore.df which lists all the files that are involved in the generation of layout.

### 8.1 Layout: For Custom Designs

Custom designs are designs that combine standard cell layout with other methods for generating layout. When specifying layout for custom designs, each area of the schematic must include its own layout atom. Here is a simple example:

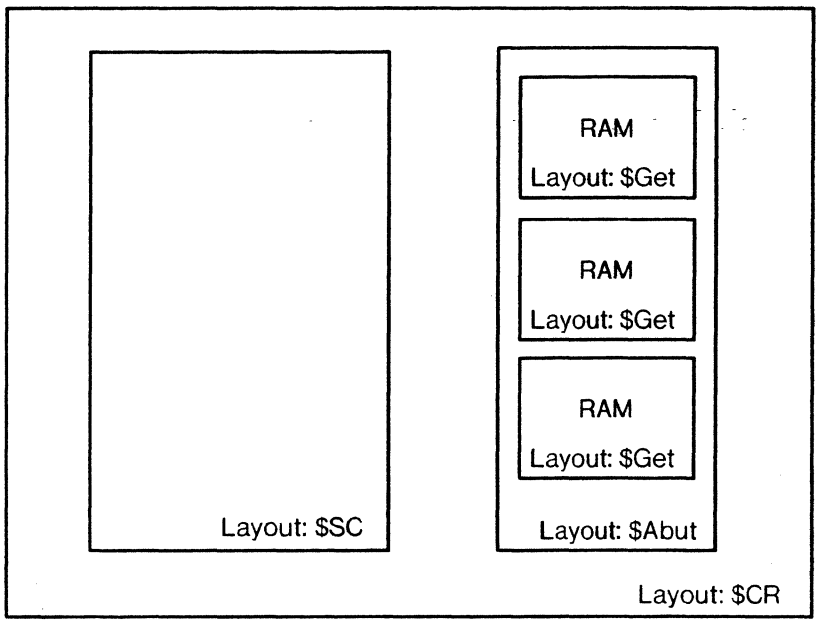



**Figure 14**: Specification of layout atoms for a cell that uses multiple layout procedures.

Here are definitions for the layout atoms used in this example:

$Get

The $Get layout atom is used whenever the layout procedure for a cell is explicitly specified. This means that $Get is the correct atom for use in the Logic Library itself and for custom layout.

$Abut

The $Abut layout atom is used when subblocks of the circuit are to be concatenated. The layout procedure, PWCore, is smart enough to figure out whether the concatenation should be in the x or y direction. At the present time, diagonal concatenation, as indicated in Figure 15, is not possible.

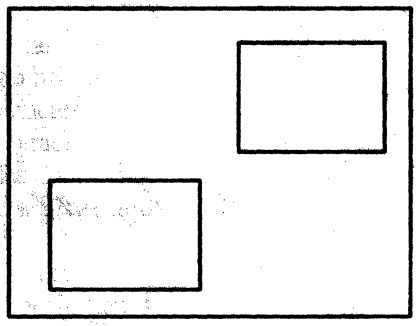


**Figure 15:** Diagonal concatenation of blocks. This configuration is NOT currently supported by PWCore.

$CR

The *Channel* layout procedure connects exactly two subcells by either a horizontal or a vertical channel.

$SC

This is the layout atom used for standard cells. This atom should only be stored one time for a given design, at the highest level of the CellType hierarchy.

To find out more about specifying layout for schematics see, [DATools]<DATools7.0>PWCore>PWCoreDoc.tioga.

## 9.0 Conclusion

This document has provided an introduction to the DATools. At this point, the paths of those who *write* design automation tools and those who *use* these tools diverge.

Tool writers might want to take a look at [DATools]<DATools7.0>DAUser>ExampleCedarProgram.Mesa and [DATools]<DATools7.0>DAUser>ExampleCedarProgramImpl.Mesa. This is a simple CEDAR program that uses a number of the important interfaces from the DATools world. It adds a final entry to the OSpace menu which reads "Sisyph -> Example." When this entry is selected, Sisyph writes the results of the extraction, that is the core data structure, to a file.

Designers are ready to begin working on a real circuit. You will undoubtedly run into problems. When you do, start by referring to the documentation. From there, ask for help. Unfortnately, most detailed information resides in a few experts heads and is not available on paper.