Magic Tcl Tutorial #4: Simulation with IRSIM

R. Timothy Edwards

Space Department
Johns Hopkins University
Applied Physics Laboratory
Laurel, MD 20723

This tutorial corresponds to Tcl-based Magic version 7.2

Tutorials to read first:

Magic Tutorial #1: Getting Started
Magic Tutorial #2: Basic Painting and Selection
Magic Tutorial #4: Cell Hierarchies
Magic Tutorial #8: Circuit Extraction
Magic Tutorial #11: Using IRSIM and RSIM with Magic

Commands introduced in this tutorial:

irsim , getnode, goto
graphnode, watchnode, unwatchnode
movenode, watchtime, unwatchtime, movetime
(plus the standard IRSIM command set)

Macros introduced in this tutorial:

(None)

1 IRSIM Version 9.6

In version 9.6, IRSIM has been redesigned to work under the Tcl interpreter, in the same manner as Magic version 7.2 does. Like Magic version 7.2, section of Tcl as an interpreter is specified at compile-time, along with various use options. The "**make**" method has been rewritten to match the one which Magic uses, so IRSIM can be compiled and installed in a similar manner:

```
make config
make tcl
make install-tcl
```

Tcl-based IRSIM, like its non-interpreter version, can be run as a standalone product, and will simulate a circuit from a `.sim` format file. However, it is specifically designed to be operated in conjunction with magic, with methods for providing feedback directly into the layout from the simulation, and vice versa. There are a number of *cross-application commands*, detailed below, which belong to neither Magic or IRSIM, but are applicable when both are running in the Tcl interpreter at the same time.

The cross-application commands highlight the usefulness of the method of compiling each application as a loadable Tcl object module.

In addition to cross-application commands, Tcl-based IRSIM allows the use of interpreter variables, conditionals, and control structures to set up detailed simulation environments. A random number generator has been added to the Tcl-based version, allowing generation of random bit vectors for statistically-based coverage of input pattern spaces.

2 Invoking IRSIM from Magic

Within the Tcl/Tk environment, IRSIM is easier than ever to invoke. For tutorial purposes, we will use the same cell used for the original Tutorial #11. Unlike the original version, Magic 7.2 requires no preparation for simulation and can operate directly off of the tutorial directory input. Start magic with the command-line

magic -w -d OGL tut11a

Note that the OpenGL interface and Wrapper environment specified above are optional, and do not affect the descriptions in this tutorial.

It is not necessary to extract! The scripts which invoke IRSIM are capable of looking for a netlist file to simulate for the currently-loaded cell. Because these exist for the tutorial cells, they will be used. IRSIM is therefore simply invoked by:

% **irsim**

You will see a slew of output that looks like the following:

```
Warning:  irsim command 'time' use fully-qualified name '::irsim::time
Warning:  irsim command 'start' use fully-qualified name '::irsim::sta
Warning:  irsim command 'help' use fully-qualified name '::irsim::help
Warning:  irsim command 'path' use fully-qualified name '::irsim::path
Warning:  irsim command 'clear' use fully-qualified name '::irsim::cle
Warning:  irsim command 'alias' use fully-qualified name '::irsim::ali
Warning:  irsim command 'set' use fully-qualified name '::irsim::set'
Warning:  irsim command 'exit' use fully-qualified name '::irsim::exit
Starting irsim under Tcl interpreter
IRSIM 9.6 compiled on Thu Mar 20 17:19:00 EST 2003
Warning:  Aliasing nodes 'GND' and 'Gnd'
/usr/local/lib/magic/tutorial/tut11a.sim:  Ignoring lumped-resistance
     ('R' construct)
```

```
Read /usr/local/lib/magic/tutorial/tut11a.sim lambda:1.00u format:MIT
68 nodes; transistors:  n-channel=56 p-channel=52
parallel txtors:none
%
```

These comments require some explanation. The warning messages all have to do with the fact that certain command names are used both by IRSIM and Magic, or by IRSIM and Tcl or one of its loaded packages (such as Tk). There are several ways to work around the unfortunate consequences of multiply defining command names, but the easiest is to make use of the Tcl concept of *namespaces*. A complete description of Tcl namespaces is beyond the scope of this tutorial; however, a simple description suffices. By prefixing a "scope" to the command, the command can only be executed when the complete name (scope plus the double colon '::' plus the command name) is entered.

In general, the EDA tools make an attempt to allow commands to be entered without the scope prefix at the command line. As long as command names are unique, this is done without comment. However, when commands overlap, the easiest solution is to require the scope prefix. Therefore, the command '**set**' would refer to the Tcl **set** command (i.e., to set a variable), while '**irsim::set**' would refer to the IRSIM command. Some attempt is made to overload commands which conflict but which have unique syntax, so that it is possible to determine which use is intended when the command is dispatched by the interpreter.

In addition to the warnings, there are a few standard warnings about global name aliases and lumped resistance, and some information about the `.sim` file which was read.

3 IRSIM Command Set

In addition to the exceptions noted above for fully-qualified namespace commands, there are several IRSIM commands which are not compatible with Tcl syntax, and these have been renamed. The old and new commands are as follows (see the IRSIM documentation for the full set of commands):

¿	savestate	save network state
¡	restorestate	restore network state
¡¡	restoreall	restore network and input state
?	querysource	get info regarding source/drain connections
!	querygate	get info regarding gate connections
	source *(Tcl command)*	source a command file

Note that the '' command is simply superceded by the Tcl '**source**' command, which is more general in that it allows a mixture of Tcl and IRSIM commands (and commands for any other loaded package, such as Magic) to be combined in the command file.

Once loaded into Tcl alongside Magic via the **irsim** command, the IRSIM commands are typed directly into the Magic command line, and will execute the appropriate IRSIM function. By repeating the contents of Tutorial #11 in the Tcl environment, this method should become clear, as will the benefits of using the interpreter environment for simulation.

To setup the simulation, the equivalent instruction to that of Tutorial #11 is the following:

% **source ${CAD_HOME}/lib/magic/tutorial/tut11a.cmd**

Note that because the **source** command is a Tcl command, not a Magic or IRSIM command, it it necessary to specify the complete path to the file, as Tcl does not understand the search path for Magic cells, which includes the tutorial directory.

As most common commands are not among the set that cause conflicts with Magic and Tcl commands, the tutorial command file loads and executes without comment.

Following the example of Tutorial #11, type **c** (IRSIM clock command) on the magic command line to clock the circuit. Values for the watched nodes, which were declared in the tutorial command file, are displayed in the console window. Likewise,

h RESET_B hold

will set the nodes **RESET_B** and **hold** to value 1.

4 Feedback to Magic

The cross-application commands reveal the usefulness of having both applications as extensions of the same Tcl interpreter.

While Magic and IRSIM are active and file `tut11a` is loaded, execute the following commands from the command line:

```
stepsize 100
watchnode RESET_B
watchnode hold
```

Note that the nodes and values are immediately printed in the magic window, making use of the magic "**element**" command. These values are persisent in the sense that they will remain through various transformations, openings, and closings of the layout window, but they are temporary in the sense that they will not be saved along with the layout if the file is written (however, this behavior can be modified).

The **watchnode** command requires no special action for placing the label elements in the layout because magic uses the labels or other node information to pinpoint a position in the layout belonging to that node, and places the label element there. It is possible to use **watchnode** with vectors. However, as no location can be pinpointed for a vector, the magic cursor box position will be used to place the label element.

Move the magic cursor box to a empty space in the layout window, and type

watchnode bits

Now move the cursor box to another empty space and type

watchtime

Now all of the simulation values of interest are displayed directly on the Magic layout.

The display of any node can be removed with the command **unwatchnode**, with the same syntax as **watchnode**, and similarly, the display of simulation time can be removed with the command **unwatchtime**.

If the position of a label is not in a good position to read, or the relative position of two labels places them on top of one another, making them difficult to read, the labels can be moved using the **movenode** command. For instance, the node `RESET_B` is not exactly on the polysilicon pad. To center it exactly on the pad, select the square pad, so that the box cursor is on it, then do

```
movenode RESET_B
```

The label will be moved so that it is centered on the center of the cursor box. The equivalent method can be applied to the time value using the **`movetime`** command.

It is not necessary to know the name of a node in order to query or display its simulation value. For instance, unexpand the layout of `tut11a.mag`, select an unlabeled node, and use a mixture of IRSIM and magic commands to watch its value:

```
box 93 -104 94 -102
select area
watchnode [getnode]
```

In this example, both the node (`bit_1/tut11d_0/a_39_n23#`) and its value are displayed. Likewise, the **getnode** command can be combined with other IRSIM commands to setup clocks and vectors from unlabeled nodes. This can be particularly useful in situations where it may not be obvious which nodes in a design need to be examined prior to running the simulation.

5 Analyzer Display

Tcl-based IRSIM has a graphical node display which is derived from functions available in the "**BLT**" graphics package. These functions are not particularly well-suited for display of logic values, and so this will probably be replaced in the future with a more appropriate interface. However, it accomplishes most of the functions of the former X11-based analyzer display.

In the Tcl-based IRSIM, no special command is needed to initialize the analyzer display. One command sets up signals to be displayed in the analyzer window. This is:

`graphnode` *name* [*row*] [*offset*]

For display of multiple signals in the window, the optional arguments *row* and *offset* are provided. Each signal which declares a new *row* (default zero) will appear in a separate graph in the display. Signals which appear in the same graph, however, may declare a non-zero *offset* which will set them at a different vertical placement on the graph, for cases in which this provides better viewing than having the signals directly overlapping.

The analyzer display updates at the end of each simulation cycle. Logic values are displayed as 0 or 1, with undefined (value 'X') values displayed as 1/2. Note that the BLT-based interface prohibits the display of multi-bit values, and only nodes, not vectors, can be passed to the **`graphnode`** command.

6 Wildcards

The original IRSIM used "wildcard" characters in the form of standard UNIX "regular expressions" to perform operations on multiple nodes with one command. Unfortunately, there was a syntactical collision between IRSIM and Magic over the use of brackets ('**[**' and '**]**'). Brackets represent groupings in regular expression syntax. However, Magic uses brackets to represent arrays of subcells. Because Tcl itself implements regular expressions in the form of the Tcl "**regexp**" command, there is a way around this problem in the Tcl version of IRSIM. IRSIM's parsing of regular expressions has been disabled. In place of it, Tcl lists may be passed as arguments to any command which previously would accept wildcard characters. In addition, Tcl-IRSIM defines a command

```
listnodes
```

which returns a Tcl list of all the nodes defined in the netlist input file. This list can be searched by Tcl regular expression commands, and the resulting sub-lists passed as node arguments to IRSIM commands. For example, the following script sets all nodes in the circuit (except for `Vdd`, which is fixed) to zero, then releases them:

```
set nl [listnodes]
l $nl
s
x $nl
```

Brackets in individual node names are treated as-is by IRSIM, as are other Magic-generated characters such as the slash, underscore, and hash mark. Note, however, that because Tcl itself defines brackets as representing command groupings which return an immediate result, the following is illegal:

```
% l multcell5_0[1,0]/a_13_n21#
invalid command name "1,0"
```

Instead, node names containing brackets should be surrounded by braces ('{' and '}'), which effectively turns a node name into a list of node names which happens to contain exactly one entry:

```
% l {multcell5_0[1,0]/a_13_n21#}
```

The Tcl versions of Magic and IRSIM are set up in such a way that when they return results containing node names, these names are automatically treated as lists. Therefore, the command

```
% select area [goto {multcell5_0[1,0]/a_13_n21#} ]
% l [getnode]
```

does not produce any error when the arrayed node name is passed to the IRSIM "**l**" command, and sets the value of the node to zero as expected. It is only when node names are entered in a script or from the command line that precautions must be taken to list-enclose names which contain brackets.

7 Scripting IRSIM Command Sequences

A consequence of placing IRSIM in an interpreter environment is the ability to use interpreter features such as variables, conditionals, and loops to set up complicated simulation environments.

8 Deterministic Bit Vector Generation

A convenience function has been added to Tcl-IRSIM to aid in generating deterministic sequences of inputs. This is the command

bconvert *value bits* [*dir*]

where *value* is an integer decimal value, *bits* is the length of the bit vector to hold the conversion, and *dir* is an optional direction flag. If *dir* is 1, then the bit vector is defined with the most significant bit (MSB) on the right. The **bconvert** command returns the string value of the bit vector containing 0 and 1 characters. For example:

```
% bconvert 20 5
10100
% bconvert 20 5 1
00101
```

9 Random Bit Vector Generation

The tutorial examples are small by design, but real systems (such as a microprocessor) are often so complex that generating and simulating an exhaustive set of all possible states of the circuit is impossible, and instead simulations rely on the generation of a set of randomly-generated inputs to test a representative set of states.

Random number generation is not a built-in feature of the Tcl language, but several open-source packages exist, one of which has been incorporated into the IRSIM 9.6 source. The pseudorandom number generator is compiled as a separate Tcl package, but is loaded by the IRSIM startup script. It contains one command, **random**, with the following arguments:

random *option*

where *option* may be one of:

-reset will cause the generator to be reseeded using current pid and current time.

-seed *n* will reseed the generator with the integer value *n*.

-integer ... will cause the number returned to be rounded down to the largest integer less than or equal to the number which would otherwise be returned.

-normal *m s* will cause the number returned to be taken from a gaussian with mean *m* and standard deviation *s*.

-exponential *m* will cause the number returned to be taken from an exponential distribution with mean *m*.

-uniform *low high* will cause the number returned to be taken from uniform distribution on *[a,b)*.

-chi2 *n* will cause the number returned to be taken from the chi2 distribution with *n* degrees of freedom.

-select *n list* will cause *n* elements to be selected at random from the list *list* with replacement.

-choose *n list* will cause *n* elements to be selected at random from the list *list* without replacement.

-permutation *n* will return a permutation of $0 \ldots n-1$ if *n* is a number and will return a permutation of its elements if *n* is a list.

The following script clocks a random serial bit vector into a state machine, assuming that **bit_in** is the node to set, and that the proper clock vectors have already been set up:

```
for {set i 0} {$i ¡ 100} {incr i} {
        if {[random] ¡ 0.5} {
                l bit_in
        } else {
                h bit_in
        }
        c
}
```