IRSIM Tutorial

EEC 116, Fall 2010

October 12, 2010

Introduction

This is an introductory tutorial for IRSIM, which is a fast switch-level simulator designed to work with an extracted Magic layout. Simulating an extracted layout allows you to check the functionality of a MOS layout at a detailed level as well as providing first-order performance measurements. To keep the example simple, the design is assumed to be a 2-input NAND gate with inputs "A, B", and output "Z". You may use your NAND from a previous homework or design a new one.

Creating a .sim file

IRSIM files are text files with the extension .sim that contain the description of an entire circuit. IRSIM files can be created by hand or extracted from Magic. Since you want to functionally verify your Magic layout, you will want to extract the IRSIM file from your .mag file. Your power nodes must be labeled "Vdd" and "Gnd" for IRSIM to recognize them correctly. Point labels are preferred over box labels for clarity. You can refer to each node in IRSIM by their Magic label name. The first step is to extract the simulation data from your Magic layout. After you have completed a design in Magic, type the following commands:

```
magic> extract all
magic> exttosim -R -L -c 20 <cell_name>.ext
```

After these commands have been executed, a .ext and .sim file will be created in your directory.

Alternatively, you can exit Magic and run
```
$ ext2sim -R -L -c 20 <cell_name>.ext
```
in the console.

Starting IRSIM

In a terminal window change directories to the location of your .sim file and type:

```
$ irsim116 nand2.sim
```

or from within Magic type:

```
magic> irsim TSMC.18
```

You should be aware that IRSIM will behave differently when run from within Magic. This tutorial assumes IRSIM is run external from Magic, which is the recommended way. After starting IRSIM it will tell you how many nodes and transistors have been recognized and then display the IRSIM prompt.

Using IRSIM

You can enter commands interactively or create a script. Scripts are saved as .cmd files and contain a list of commands just as they are typed at the IRSIM prompt. Here is an example of an IRSIM script which can also be done interactively. The `irsim>` prompt precedes commands that are entered interactively. Text preceded by the vertical bar | is output from IRSIM. Instructional comments are preceded by #.

```
# The basic idea of IRSIM is you tell it which nodes to pull high, low,
# and tristate. Then you tell IRSIM to run the simulation for a certain
# period of time.  This period of time is the step size. The 'stepsize'
# command tells IRSIM what the step size should be, the default is 10ns.

irsim> stepsize 50

# The 'w' command tells IRSIM to watch the nodes change. The command
# below tells it to watch the nodes A, B and Z. IRSIM displays the
# nodes in the opposite order of that set by the command, therefore
# the output order will be A B Z. This is just a matter of personal
# preference though. Enter the nodes in any order you like.

irsim> w Z B A

# 'd' displays all the nodes that are being watched. You can also enter
# in something like 'd A' which tells IRSIM to only display the node A.

irsim> d
| A=X B=X Z=X
| time = 0.00ns

# At time zero, the values for the nodes are all undefined. The 'l'
# command forces the nodes to a logic value of 0.

irsim> l A B

# 's' simulates for the period of time previously defined by the
# 'stepsize' command. IRSIM displays the value of each node being
# watched after each step.

irsim> s
| A=0 B=0 Z=1
| time = 50.00ns

# The 'h' command sets the following nodes to a logic value of 1.

irsim> h A B
irsim> s
| A=1 B=1 Z=0
| time = 100.00ns

# The 'path' command shows the critical path for the last node
# transition. The output shows that an input node A changed to
# logic 1 at time = 50.00 ns. Then node Z changed to 0 at
# time = 50.01 ns. Therefore it took 0.01 ns to go from high to
# low for the given input change.
```

```
irsim> path Z
| critical path for last transition of Z:
|   A -> 1 @ 50.00ns , node was an input
|   Z -> 0 @ 50.01ns   (0.01ns)


# If you have a long list of nodes, it can be tiresome to keep using
# the l and h commands to set their logic values. The 'vector' command
# lets you group nodes together so you can set them all quickly. The
# command below tells IRSIM to group the nodes A and B into a vector In.
# The first node will be the MSB.

irsim> vector In B A

# The 'setvector' command tells IRSIM to set the value of a vector.
# The first command below sets the vector In to 00, therefore
# A=0 and B=0. The following commands demonstrate how you can create a
# truth table using the vector In.

irsim> setvector In 00
irsim> s
| A=0 B=0 Z=1
| time = 150.00ns
irsim> setvector In 01
irsim> s
| A=1 B=0 Z=1
| time = 200.00ns
irsim> setvector In 10
irsim> s
| A=0 B=1 Z=1
| time = 250.00ns
irsim> setvector In 11
irsim> s
| A=1 B=1 Z=0
| time = 300.00ns

# To check a value, you can use assert commands.
# If the assert passes, no output is given.

irsim> assert Z 0
irsim> assert Z 1
assertion failed on 'Z' 1 (0)

# The 'listnodes' command will print out a list of nodes which are
# available for simulation. This is useful for checking label placement
# and node conductivity. Nodes ending with a # are usually internal
# nodes. In the case of a NAND gate this is the node between the NMOS
# transistors.

irsim> listnodes
| A B Z Gnd Vdd a_n4_n11#
```

```
# The 'querygate' command shows the gate connections for a given node,
# and is also a good debugging technique. The line “p-channel A=1 Vdd=1
# Z=0” means that for the PMOS transistor, A is tied to the gate while
# the source and drain are tied to Vdd and Z. The line “n-channel A=1
# Gnd=0 a_n4_n13#=0” means that for the NMOS transistor, A is tied to the
# gate while the source and drain are tied to Gnd and the internal node.
# '!' is a shorthand command for 'querygate'.

irsim> querygate A
| A=1 [NOTE: node is an input] (vl=0.30 vh=0.80) (0.0001 pf) affects:
| p-channel A=1 Vdd=1 Z=0 [49.8K, 38.8K, 143.3K]
| n-channel A=1 Gnd=0 a_n4_n11#=0 [21.3K, 47.8K, 15.8K]

# To restart a simulation, you can type in 'start'.  This actually restarts
# irsim.  Another option is to use the 'back time' command to move back
# to the start of simulation time.
```

Using the analyzer window

The analyzer window is an useful graphical logic analyzer used to debug the operation of a design. Signals can take on the values {0, 1, X}. As you step the through the simulation, you can see the value of any signal in the analyzer window as they change. You can view the analyzer window using the command `'analyzer'` or `'ana'`. You can also start the analyzer by following the command with the names of available nodes. If the window is already visible this will append the nodes to the list.

```
irsim> analyzer A B Z
```
or
```
irsim> ana A B Z
```

When the window is visible you can also add available nodes by using "Manage Traces"under the Window menu. To see the time for an event, and the values at that time, click inside the analyzer window to show the vertical time line. The time corresponding to the vertical line is displayed above. To find the duration of an event you can left click to place the cursor then hold down right click to highlight the time interval in red. IRSIM will display the times corresponding to each cursor and the interval between them.

Simulating Clock Signals

To simulate the clock signal we need to first define which node is the clock and then define the sequence which should occur during a single clock period. This can be done using the `'clock'` command.

```
irsim> clock clk 0 1
```

Once the clock sequence has been defined we can step the simulation by a multiple of the clock period using the `'c'` command followed by the number of periods. Using the `'p'` command we can simulate a signal clock phase, or half-period (equal to the stepsize). Both clock commands will toggle the clock node where simply using the `'s'` command will keep values the same.

Additional notes for IRSIM

You have seen the commands `'l'` and `'h'` to set nodes to logic 0 and logic 1. The `'x'` command will effectively put a node into High-Z for simulating tri-state buffers. You can also use the `'u'` command to set a node to unknown.

```
irsim> x A
irsim> u B
```

You can run a .cmd file from the command line by typing a dash in front of the filename:

```
$ irsim116 nand2.sim -nand2.cmd
```

To run a .cmd file with IRSIM (i.e. a script of commands) you can do the following:

```
> irsim116 <cell_name>.sim -<cmdfile>.cmd
```

Or, from within IRSIM, you can use the @ command:

```
irsim> @ <cmdfile>
```

To include multiple cell, you can give multiple .sim files as arguments in the command line. However, it is recommended to flatten to a single cell and irsim that singular cell to prevent problems with the nodes.

When running IRSIM from within Magic some additional commands are available. The commands `'watchnode'` and `'watchtime'` allow you to view the value of nodes inside your magic layout. You can use `'unwatchnode'` and `'unwatchtime'` to hide the values. To move the values create a point with the box and use the commands `'movenode'` and `'movetime'`.

```
magic> watchnode A B Z
magic> watchtime
```

Tutorial from http://www.ece.ubc.ca/~elec479/irsim_tut.pdf

2004	Additions and localizations by Bevan Baas and Omar Sattari
2006/05/28	Rewritten and updated for IRSIM 9.7 by Eric Work
2010/01/27	Updated by Trevin Murakami
2010/10/01	Yet another slight update by Trevin Murakami due to switch to Tcl