IRSIM Tutorial (version 2.1)

1.0 IRSIM the switch-level simulator

This is a simple tutorial on irsim. It is not meant to replace the irsim manual which contains much more information. Try `man irsim` for information on all the commands that `irsim` supports. `irsim` is a switch-level simulator, in the sense it models the circuit at the level of transistors.

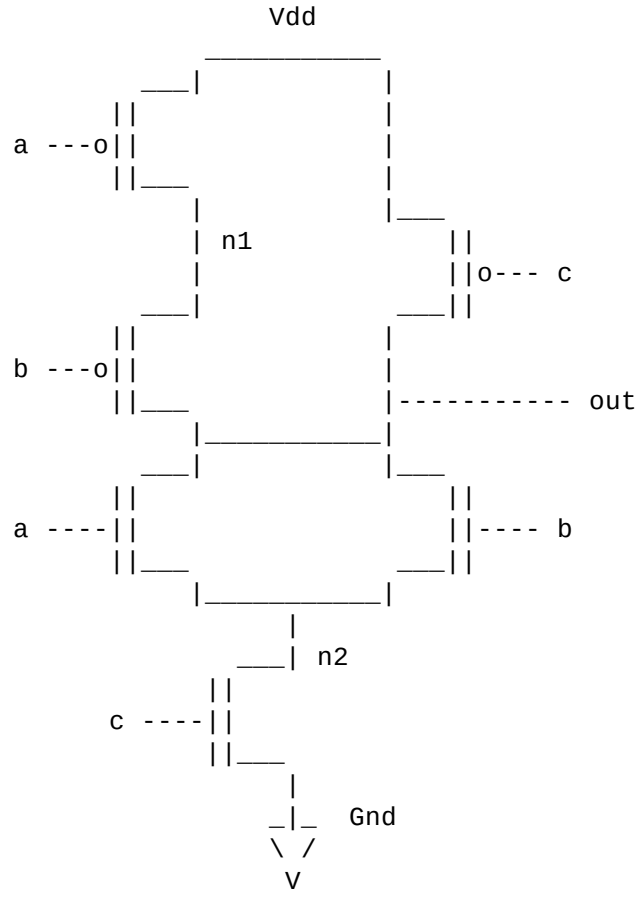
2.0 Creating a .sim file

Typically we will first use Magic to layout a circuit and then extract the circuit description and analyze it with irsim. Hence we will not typically create **.sim** circuit description manually, as is described below.

Manually creating a circuit description (.sim file) for irsim can be useful in some cases. For example it permits a transistor level design to be tested quickly, before a layout is generated.

Irsim files are just text files that contain information about a circuit. They have the extension .sim. The first time you use irsim, you will probably have to create the .sim file yourself using your favorite text editor.

Below is a CMOS circuit to implement *~a~b + ~c* where *~a* means the inverse of *a*.



The first step in creating a **.sim** file for this circuit is to label all the nodes. Label power *Vdd* and ground *Gnd*

Irsim is not case sensitive so *vdd* and *gnd* will work also. You can label the other nodes anyway you want. Use any method you wish. For the circuit above, the inputs are labeled *a*, *b*, *c* and the output node is labeled out. Other internal nodes are labeled *n1* and *n2*. When using irsim, it is helpful to have the labeled circuit schematic available so that you know the names of the nodes that you want to probe. The tut_irsim_gate.sim file for the above circuit is shown below.

```
|units: 100  tech: scmos
|
|type  gate     source  drain   length  width
|----  ----     ------  -----   ------  -----
p       a       vdd     n1      2       4
p       b       n1      out     2       4
p       c       vdd     out     2       4

n       a       out     n2      2       4
n       b       out     n2      2       4
n       c       n2      gnd     2       4
```

The first four lines are comments. Any line that begins with the vertical bar '|' is a comment. This first line says that the technology used is scalable cmos.

Transistors are specified as follows:

```
|type gate source drain length width
|---- ---- ------ ----- ------ -----
```

where type is **p** for pmos and **n** for nmos. The gate, source, and drain refer to the terminals of the transister. This is why you need a labeled circuit schematic to create a **.sim** file by hand. It is important that you get the connections correct. The length and width of the transistor is both in microns.

The first transistor specified in the example above is the top-left pmos transistor in the corresponding schematic. Study the rest of the irsim file and the circuit. It is easy to see how the two match. You can also enter in resistors and capacitors. See the irsim manual for the proper format to create these elements. Considering just transistors, you are now ready to run irsim.

3.0 Starting IRSIM

At the unix prompt type

```
irsim tut_irsim_scmos2um.prm tut_irsim_gate.sim
```

where
tut_irsim_gate.sim is the simulation file and
tut_irsim_scmos2um.prm is the parameter file of a 2 micron process technology.

irsim will tell you how many transistors you have and then display the irsim prompt shown below.

```
IRSIM>
```

3.1 Running IRSIM

You can enter in commands interactively or you can run a script. Scripts are just the .cmd files and they

contains commands that are exactly what you would type in at the irsim prompt. (See section 3.3 below).

Here is an example irsim session. The IRSIM> prompt precedes commands that were entered interactively. Text preceded by the vertical bar | is output from irsim.

```
IRSIM> stepsize 50
```

The basic idea of irsim is that you tell it which nodes to pull high, which nodes to pull low, which nodes to tristate, and then you tell irsim to run the simulation for a certain period of time. This period of time is the stepsize. The above command tells irsim that the stepsize is 50ns. The default stepsize = 100 nanoseconds.

```
IRSIM> w out c b a
```

The command 'w' tells irsim to *watch* the following nodes. So the above command tells it to watch the nodes *out*, *c*, *b*, and *a*. Irsim displays the nodes in the reverse order in the above command. Therefore the output order will be *a b c out*. This is just a matter of personal preference though. Enter in the nodes in any order you like.

```
IRSIM> d
```

d *displays* all the nodes that are being watched. You can also enter in something like 'd a b' which tells irsim to only display nodes *a* and *b*

```
| a=X b=X c=X out=X
| time = 0.0ns
```

at time zero, the values of the nodes are all undefined

```
IRSIM> l a b c
```

The l command forces the nodes to a logic *low* value of 0 The above command sets nodes a b and c to logic 0

```
IRSIM> s
```

s tells irsim to *simulate* for a certain period of time previously defined by the stepsize command. The default value is 100 ns, but we set it to 50ns above.

```
| a=0 b=0 c=0 out=1
| time = 50.0ns
```

Irsim displays the values of the nodes after each step because of the previous d command. The current time is also displayed. Note that time = 50 ns. This is the current simulation time now.

```
IRSIM> h c
```

h sets the following nodes to a logic *high* value. Therefore the above command sets node *c* to logic 1.

```
IRSIM> s
| a=0 b=0 c=1 out=1
| time = 100.0ns
```

step again

```
IRSIM> h b
IRSIM> s
| a=0 b=1 c=1 out=0
| time = 150.0ns

IRSIM> path out
| critical path for last transition of out:
| b -> 1 @ 100.0ns , node was an input
| out -> 0 @ 100.1ns (0.1ns)
```

The path command shows the critical path for the last transition of a node. The output shows that an input node 'b' changed to logic 1 at time = 100 ns. The node 'out' then transitioned low at time = 100.1 ns. Therefore it took .1 ns to go from high to low for the given input changes.

You can look back at previous commands to verify that this is indeed what happened.

Sometimes you may want to find out what the worst case Tplh or Tphl is. The path command helps you find this number. You do this by setting the circuit to some state and then force inputs to change that will cause a transition at the output. Some intelligence is required to figure out what the worst state is and what combination of input changes will cause the slowest output transition.

If you have a long list of inputs, it can be tiresome to keep using the l and h commands to set the logic values. The 'vector' command lets you group inputs together so that you can set them all quickly

```
IRSIM> vector in a b c
```

The above command tells irsim to group the nodes a b and c into a vector called 'in'

```
IRSIM> set in 000
```

This command tells irsim to set the vector in to 000 Therefore, node a = 0, node b = 0, and node c = 0

The following commands demonstrate how you can create a truth table using the vector 'in'. Note that you can do this much faster this way than by using the commands l and h.

```
IRSIM> s
| a=0 b=0 c=0 out=1
| time = 200.0ns
IRSIM> set in 001
IRSIM> s
| a=0 b=0 c=1 out=1
| time = 250.0ns
IRSIM> set in 010
IRSIM> s
| a=0 b=1 c=0 out=1
| time = 300.0ns
IRSIM> set in 011
IRSIM> s
| a=0 b=1 c=1 out=0
| time = 350.0ns
IRSIM> set in 100
IRSIM> s
| a=1 b=0 c=0 out=1
| time = 400.0ns
```

```
IRSIM> set in 101
IRSIM> s
| a=1 b=0 c=1 out=0
| time = 450.0ns
IRSIM> set in 110
IRSIM> s
| a=1 b=1 c=0 out=1
| time = 500.0ns
IRSIM> set in 111
IRSIM> s
| a=1 b=1 c=1 out=0
| time = 550.0ns
```

irsim has a built in graphical logic analyzer which runs under X to let you view waveforms. The 'analyzer' command sets up the analyzer window

```
IRSIM> analyzer a b c out
```

This tells irsim to display the nodes a b c and out in the analyzer window

3.2 Using the analyzer window

Go ahead and experiment with the analyzer window. It's pretty easy to use. Here are some of the things that you can do.

3.2.1. Print the waveforms to a postscript file

Go to the 'print' menu and select 'file'. You will be asked for a filename. Hitting return selects the filename shown in (). Once you have the postscript file, you can print it out on one of the network printers using lp -d spot file.ps.

3.2.2. Setting the width

3.2.2a. You can set the width of the view from either the menus or the slider on the bottom. From the menus, select 'window' and then 'set width'. You will be asked to enter the width in time steps. Use the 'move to' option under the 'window' menu to move to the desired time.

You can also use the 'zoom' menu to zoom in and out.

3.2.2b. The best way to get a feel for the slider on the bottom is to play with it. Using the left mouse button expands the view or shortens the view to the left depending on where you click. Clicking using the right mouse button expands or shortens the view to the right depending on where you click. You can grab the slider with the middle button and move it around.

3.2.3. Changing the order of the nodes

If you don't like the order that the nodes are displayed, you can click inside the name with the left mouse button, and drag it to a new place. Play around with this to see how it works.

3.2.4. Finding the time of an event

You can click inside the analyzer window where the waveforms are to get a vertical line. The time where this vertical line is is displayed above. This is useful if you want to find out when a rising edge occurs. The resolution is best when you are zoomed in. You have seen the commands 'l' and 'h' to set nodes to logic 0 or logic 1. The 'x' command will effectively put a node into tristate. It does this by taking the node off the input list.

The following is an example of using the x tristate command `IRSIM> **x bus**`

puts the node bus into tristate. An application of this is simulating a register. Assume you have a one bit register that is connected to a line called bus and is operated as follows. To write a value in your register, you need to drive the value onto the bus and then set a control line write to high. To read the register, you first need to stop driving the bus. Then set a control line read to high. The irsim commands you need to issue are shown below.

```
IRSIM>  l read write            //  set read and write control lines low
IRSIM>  h bus                   //  let's write a 1 into the register
IRSIM>  s                       //  step
IRSIM>  h write                 //  write into the register
IRSIM>  s
IRSIM>  l write                 //  stop writing
IRSIM>  s
IRSIM>  l bus                   //  set bus to 0 so we can see the bus
                                //  transition from 0 to 1 when we read
                                //  the register
IRSIM>  s
IRSIM>  x bus                   //  stop driving the bus so we don't get
                                //  bus contention when we read the
                                //  register
IRSIM>  s
IRSIM>  h read                  //  read the register;  you should see a
                                //  0 to 1 transition
```

Irsim lets you keep a log of your commands and the outputs. To start recording, type

`IRSIM> **logfile anyfilename**`

Anything that you see in irsim from now on will be into the file 'anyfilename'

When you're done recording, type the following. `IRSIM> **logfile**`

Quitting IRSIM `IRSIM> **exit**`

3.3 Automation through .cmd files

Any irsim interactive command is valid. Use your favorite text editor to create this file. An example is shown below for the previous circuit.

```
stepsize 50
analyzer a b c out
vector in a b c
set in 000
s
set in 001
```

```
s
set in 010
s
set in 011
s
set in 100
s
set in 101
s
set in 110
s
set in 111
s
```

This example runs through all the possible combinations of inputs. The waveforms will be displayed in an analyzer window. You can run a .cmd file from the command line or within irsim. If the above file is called example.cmd, then at the Unix shell prompt, type

```
irsim scmos2um.prm circuit.sim -example.cmd
```

this will run the command file on circuit.sim

IRSIM commands

```
stepsize [n]                    set simulation stepsize to n ns
s        [n]                    simulate for n nanoseconds (default= stepsize)

d        [node]                 Display status of concerned nodes
w        [-]node                Watch a new node [or -remove a node] from displa

h        node1 node2 ...        High: Set list of nodes continuously to logic 1
l        node1 node2 ...        Low:  Set list of nodes continuously to logic 0
u        node1 node2 ...        Undefined: Set list of nodes to "X" (undefined)
x        node1 node2 ...        Tristate:  Stop setting the list of nodes

vector   label node(s)          define bit vector
set      label bits             Set the bits to label

ana      node1 node2 ...        (or analyzer) display nodes in analyzer window
clear                           Clear waveform viewer display

flush    [time]                 flush history upto time (default : now)
logfile  file.log               Turn logfile on
logfile                         Turn logging off
q                               quit
```

IMPORTANT. When you set a node high or low using the h or l commands, the node keeps being set to high or low (no matter what the circuit is trying to do to the node!) until you use the x command to stop setting the node.

Vectors

Since nodes typically are grouped into vectors, it is usually easier to look at *N*-bit quantities as single vector entities. The following commands can be used to define vectors and display them.

`vector` *name node1 node2* ...	define a new vector called *name* consisting of the list of nodes
`d` *name*	display a vector as an array of bits
`ana` *name*	add vector *name* to the waveform viewer
`set` *name value*	set the bits of vector *name* using the binary string *value*

To set a vector to a hexadecimal number, use:

```
set name %xhexstring
```

For instance, you can now say: `set Va %xff` if `Va` is an 8-bit vector instead of `set Va 11111111`. The prefix `%x` says that what follows is a hex constant.

A useful shortcut to defining arrays as long vectors is to say:

```
IRSIM> vector name a.b[{31:0}]
```

Clock Definition

The standard clock definition is shown below:

```
IRSIM> vector CLOCK CLK _CLK
IRSIM> clock CLOCK 01 10
```

The first line defines `CLOCK` to be a vector of two signals: `CLK` and `_CLK`. These signals are defined as globals in the file. The second line states that a single cycle of the clock consists of setting the vector first to `01` and then to `10`. Once this clock is defined, you can run the simulation for a clock step by saying:

```
IRSIM> c
```

The following runs the simulation for 10 cycles:

```
IRSIM> c 10
```

Acknowledgements

Original version by Williams, 11-apr-95. Downloaded from The IRSIM Tutorial version 2.0 Downloaded from www-leland.stanford.edu/class/ee272/doc/faq/irsim/ Modified by Fred DePiero at CalPoly 10/21/97. Also, EECS 314 Instructor Rajit Manohar at Cornell University. Finally,Modified by Francis G. Wolff at CWRU 7/18/00.