Using Verdi to Generate cTags databases for Vi and Emacs

Via VC Apps and the Verdi Knowledge Database

David Carson

Huawei Technologies Canada

April 21, 2017

Canada

Agenda

Verdi Debug Platform

Introduction to Tagging

Building Tagging Databases with a VC App

Tagging Demo

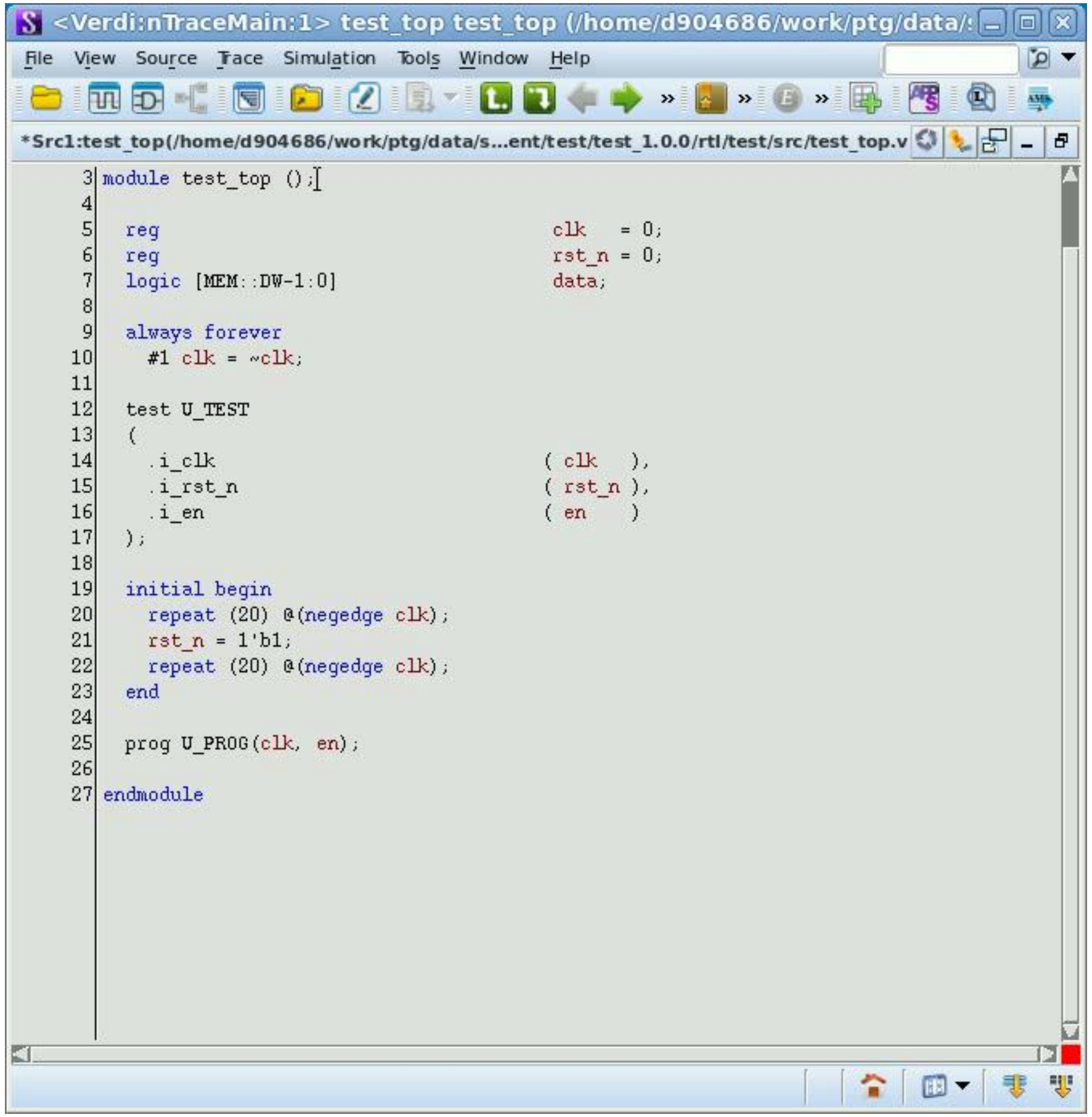
Verdi Debug Platform

The Power of the Verdi Debug Platform

wicked design exploration

- Hierarchical source code browsing
- Scope aware
- Hyperlink to definition locations
- Parameter values
- Only considers compiled code

Project Browsing with Verdi



<Verdi:nTraceMain:1> test_top test_top (/home/d904686/work/ptg/data/
File View Source Trace Simulation Tools Window Help
*Src1:test_top(/home/d904686/work/ptg/data/s...ent/test/test_1.0.0/rtl/test/src/test_top.v

```
module test_top ();

  reg                                   clk   = 0;
  reg                                   rst_n = 0;
  logic [MEM::DW-1:0]                   data;

  always forever
    #1 clk = ~clk;

  test U_TEST
  (
    .i_clk                             ( clk   ),
    .i_rst_n                           ( rst_n ),
    .i_en                              ( en    )
  );

  initial begin
    repeat (20) @(negedge clk);
    rst_n = 1'b1;
    repeat (20) @(negedge clk);
  end

  prog U_PROG(clk, en);

endmodule
```

Project Browsing with Verdi

The Power of the Verdi Debug Platform

wicked design exploration

- Hierarchical source code browsing
- Scope aware
- Hyperlink to definition locations
- Parameter values
- Only considers compiled code

It seems to know everything…

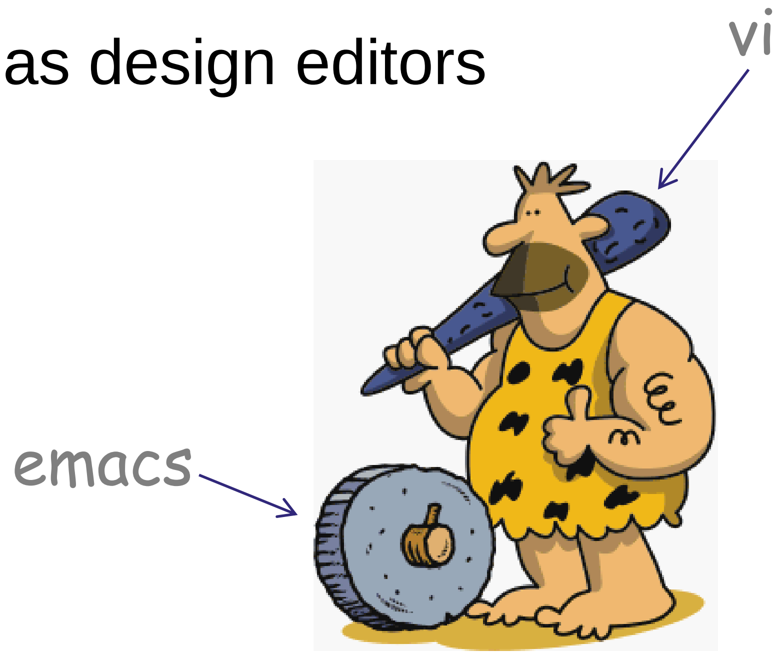
…too bad it’s not code editor

tagging

Now If It Was An Editor Too…

and as good as vi and emacs

- For decades designer have been using vi and emacs as design editors
  - They are ~~ancient~~ venerable code editors
  - They are powerful
  - They are extensible
  - They are ubiquitous
- But they are also (usually) ***not*** design aware



What if we could vi and emacs – which are powerful code editors - into light weight design exploration tools

Tagging in vi and emacs

follow that identifier!

- A “tag” is a SystemVerilog identifier

```
reg    foo;         <- definition
assign foo = 0;     <- identifier
```

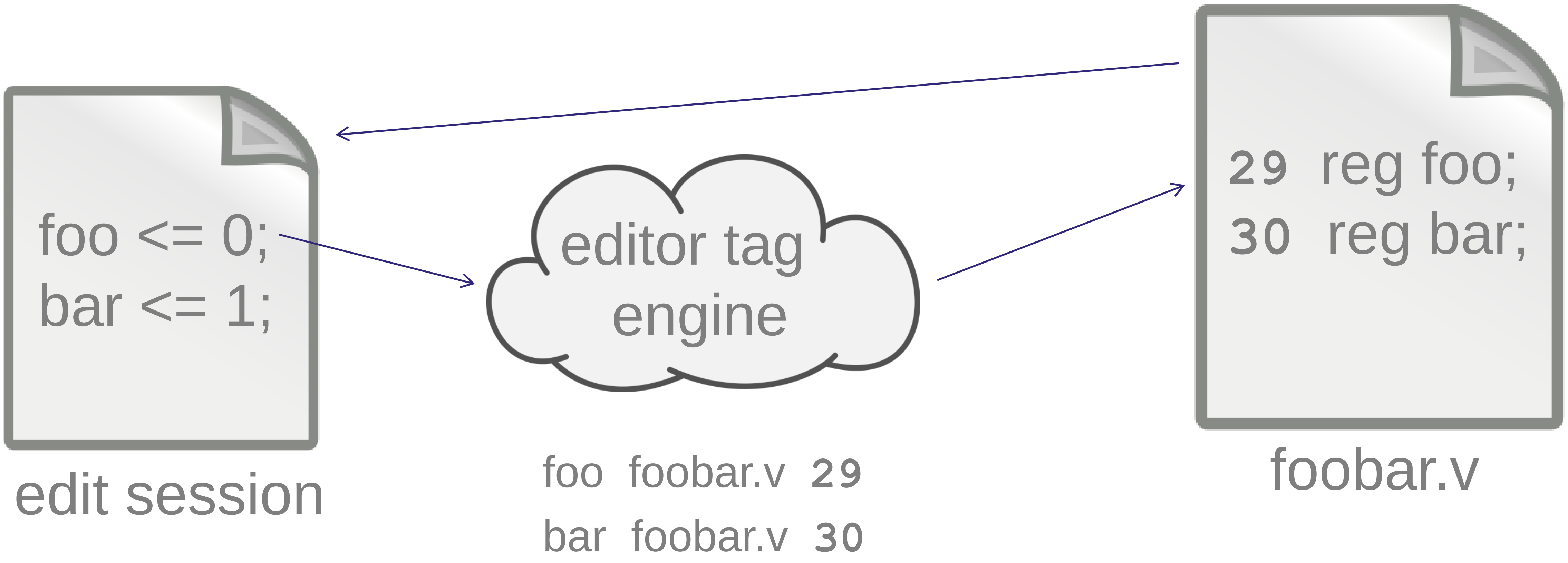
- “Tagging” jumps to the “definition” of the identifier

- Editors support tagging by using a “tag database”

Where the Heck is **foo** Defined?

tag files and their secrets

- vi and emacs use *Tag files* to track down where identifiers are defined
- Requests are made to the editor to find a definition



Hey… but that’s not new

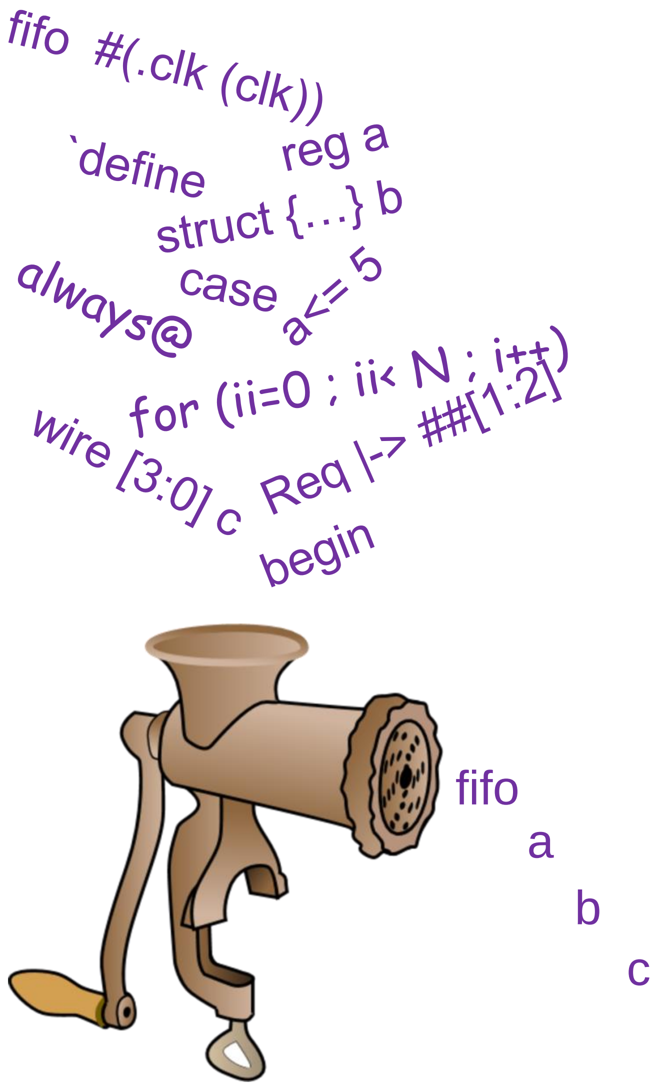
tagging has been around forever and its great

- Vi, Emac and ctags have been around since the mid 1970s
- ctags was re-written in the mid 90s and re-released it as exuberant ctags
- A new release of Exuberant cTags has not come out since 2009
- Gnu Global is the latest tagging platform for generating tagging information

What Tagging is Not

…but it's still pretty great

- Taggers are light weight language parsers
- Tagging does not pre-process
- Taggers are not compilers



cTags, Verilog and SystemVerilog

a match made in you know where…

- Original language parser based on C/C++
- Add in verification, low level net-list and timing constructs
- Not a broad user base
- No real attempts to create a more complete parser

Another way to create tags

Ahhhhh, but didn't you Mentioned Verdi?

what does Verdi have to do with any of this…

- cTags is ok…
- Gnu Global is, well, new…

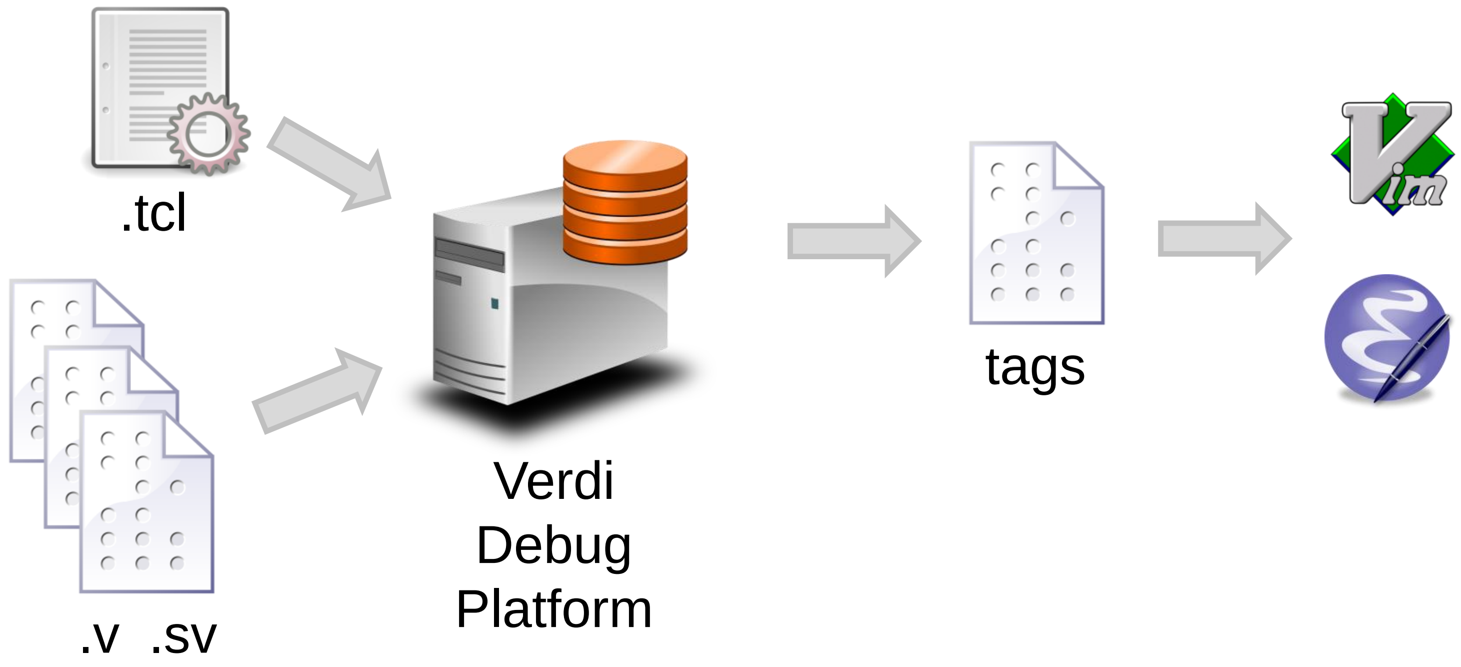
- What is an ASIC designer to do?
- Perhaps Verdi can help…

- The Verdi Debug Platform gives designers access to detailed information about a compiled design

How Exactly Does That Work?

ctags with Verdi … *really*?

- Verdi is called in batch mode with an NPI based App tcl script
  - Design loaded with a manifest or file list
  - Design is compiled in Verdi into *Knowledge Data Base*
  - The tcl script drives Verdi to parse data base for all relevant identifier information
  - Identifier information written to tag file for use by Vim and Emacs



The Good, the Bad

and it’s not that bad…

- The Good: *Verdi is a true compiler*
  - Based on fully elaborated and compiled design
  - Only those identifiers associated with the compiled design are tagged

- The Bad: *Verdi is a true compiler*
  - It’s slow
  - No real time generation of tags

No Ugly, Just More Good

taggable objects - a very comprehensive list

- SystemVerilog identifier types

ArrayNet	GenVar	ModuleArray	ShortIntTypespec
ArrayTypespec	GenScope	Net	ShortIntVar
ArrayVar			ShortRealTypespec
BitTypespec			ShortRealVar
BitVar			StringTypespec
ByteTypespec			StructNet
ByteVar			StructTypespec
ClassDefn			StructVar
ClassTypespec			Task
ClassVar			TypeParameter
ClassObj			TypePattern
Constant			TypespecMember
EnumNet			UnionTypespec
EnumTypespec	Modport	RefObj	UnionVar
EnumVar	Module	Reg	VirtualInterfaceVar

Sorry, What does good mean?

tagable objects - a few examples

module definitions, IO declarations

user defined types

all types of variables

genvars, scoping labels

classes, tasks and functions

classes, tasks and functions

programs, packages, parameters, constants

```
module test ( input logic i_clk);

  struct { logic a; } struct_a;
  union  { logic b; } union_b;
  enum   { CC       } enum_c;

  reg                 reg_var;
  wire                wire_var;
  logic               logic_var;
  integer             integer_var;

  for (genvar ii=0 ; ii<MOD_A::MOD_A_N ; ii++) begin: for_label
    mod_a U_MOD_A ( i_clk );
  end

  class C;
    int x;
    task set (int i);
      x = i;
    endtask
    function int get;
      return x;
    endfunction
  endclass

  progA U_PROG_A (i_clk, resetA);

endmodule

program progA (input wire i_clk, output logic reset);
  initial begin
    @ (posedge clk);
    reset   = 1;
  end
endprogram

package PKG
  parameter             PARM=0;
endpackage
```

VC Apps in the Verdi Debug Platform

the Native Programming Interface (NPI)

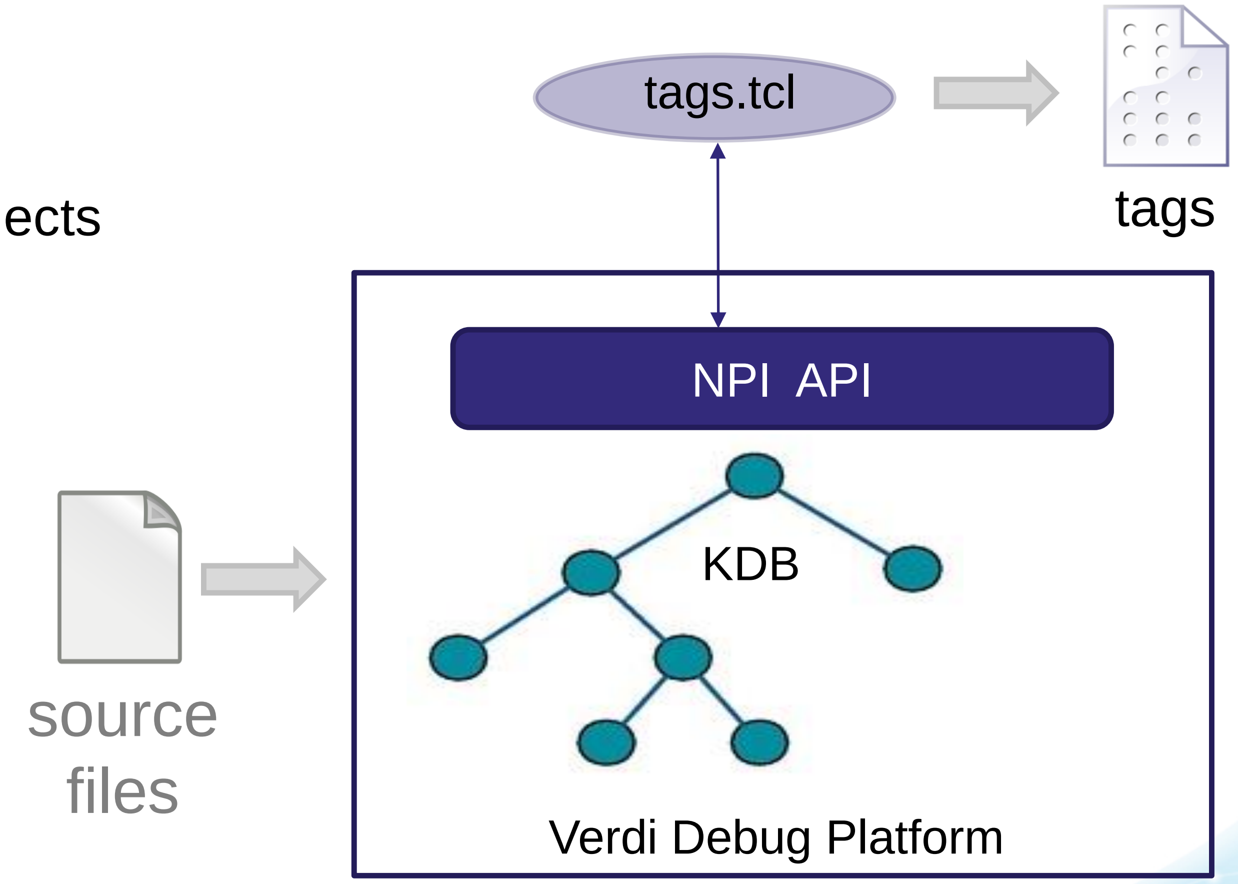
- NPI provides an Interface layer to let users access the Knowledge Data Base of compiled designs
- The NPI Language Model treats HDL language constructs as Objects
- VC Apps may be added to the Verdi Apps Toolbox

tags.tcl

a VC app

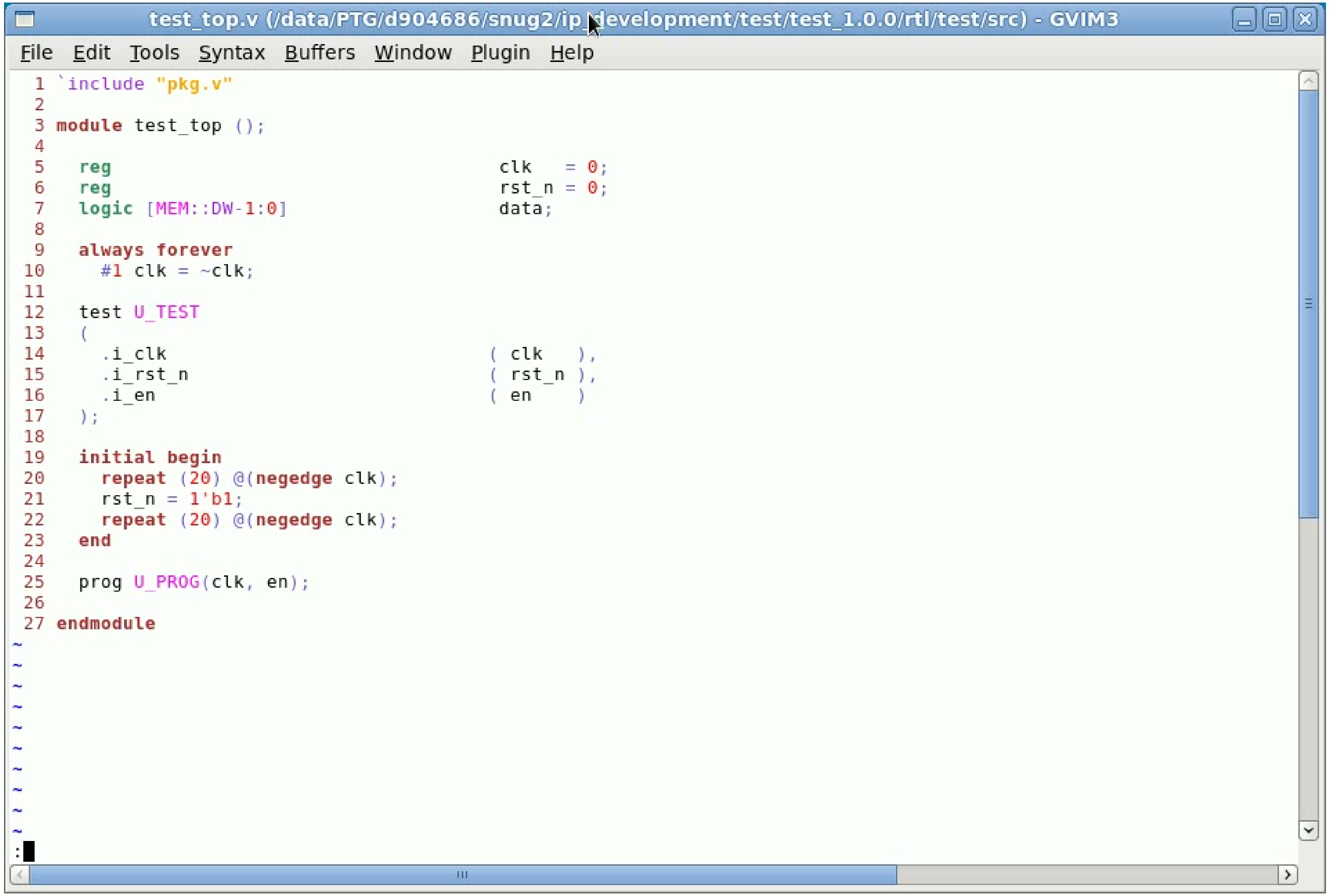
- the *tags* VC app creates a list of all SystemVerilog identifiers for a given design

1. load design
2. 1st pass - identify compile-able objects
3. 2nd pass - record all identifiers
4. create vi tags database
5. create emacs data base



Vi Based Tagging Demo

Design Browsing in Vi


test_top.v (/data/PTG/d904686/snug2/ip_development/test/test_1.0.0/rtl/test/src) - GVIM3
File Edit Tools Syntax Buffers Window Plugin Help

```
`include "pkg.v"

module test_top ();

  reg                                       clk   = 0;
  reg                                       rst_n = 0;
  logic [MEM::DW-1:0]                       data;

  always forever
    #1 clk = ~clk;

  test U_TEST
  (
    .i_clk                                 ( clk   ),
    .i_rst_n                               ( rst_n ),
    .i_en                                  ( en    )
  );

  initial begin
    repeat (20) @(negedge clk);
    rst_n = 1'b1;
    repeat (20) @(negedge clk);
  end

  prog U_PROG(clk, en);

endmodule
~
~
~
~
~
~
~
~
~
~
:
```

Improvements

there is always more to do

- tags with design scope could be added to tag files
  - if editor could identify scope it would allow more accurate tagging
- exuberant ctags information could be added to tag files
  - for use with plug ins
- tcl script could easily be converted to C
  - would improve performance

That’s Fantastic… But Where Can I Get It?

pssst… looking for a cheap app?

- Synopsys VC Apps are freely exchanged via the:

VC Apps Exchange

www.vc-apps.org

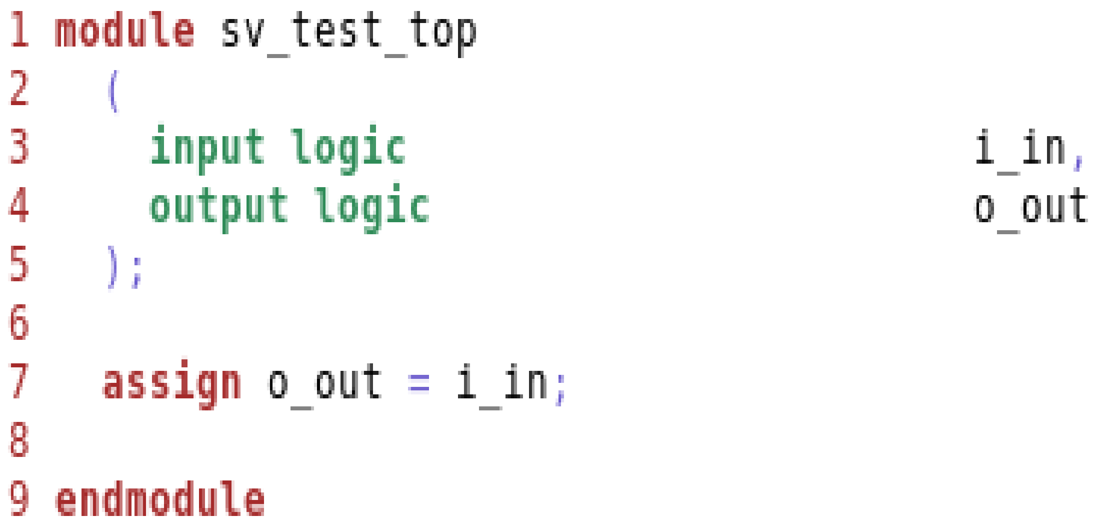
- Look for a script called:

tags.tcl

Thank You

Appendix I
a vi tag database

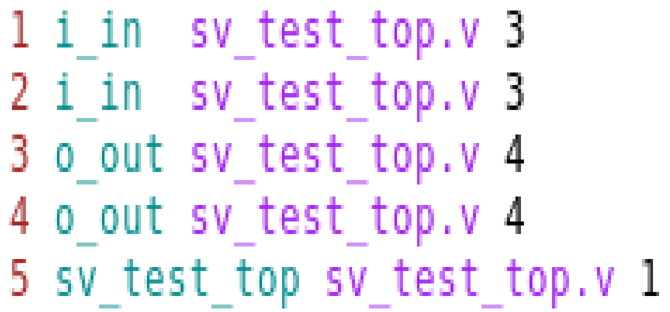
- identifier, file name and line number
- paths are absolute
- emacs format is more complex



```
module sv_test_top
  (
    input logic                              i_in,
    output logic                             o_out
  );

  assign o_out = i_in;

endmodule
```

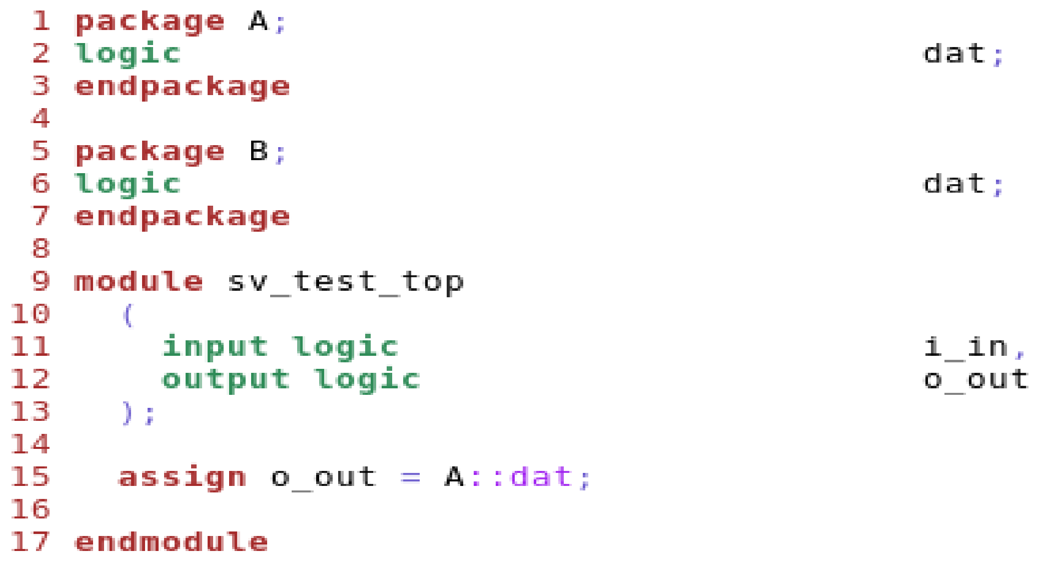


```
i_in  sv_test_top.v 3
i_in  sv_test_top.v 3
o_out sv_test_top.v 4
o_out sv_test_top.v 4
sv_test_top sv_test_top.v 1
```

Appendix II

multiple matches in tag database

- Multiple tags may exist in code
- There will be multiple matching tags
- Editor will present user with options



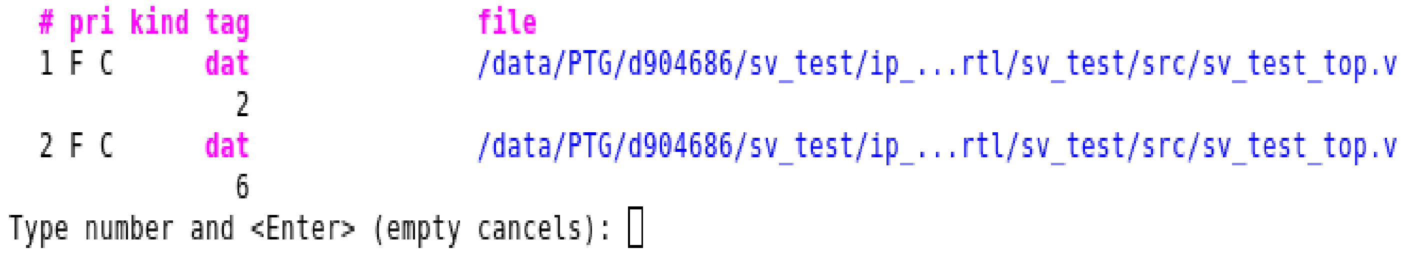
```
package A;
logic                                   dat;
endpackage

package B;
logic                                   dat;
endpackage

module sv_test_top
  (
    input logic                         i_in,
    output logic                        o_out
  );

  assign o_out = A::dat;

endmodule
```

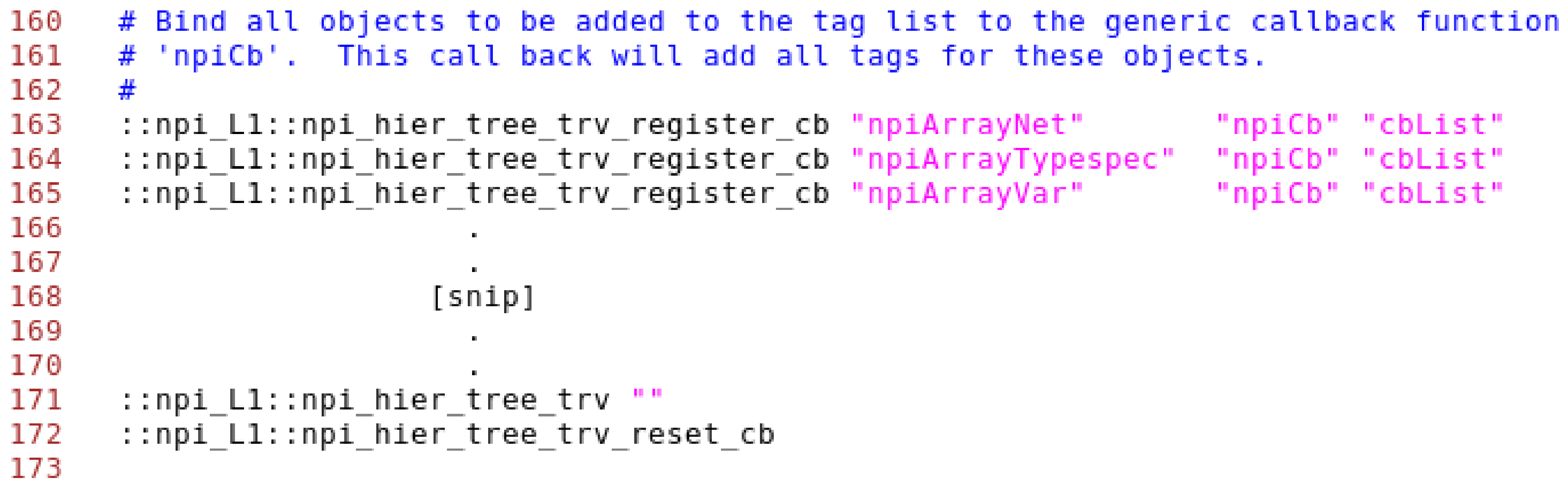


```
 # pri kind tag              file
 1 F C      dat              /data/PTG/d904686/sv_test/ip_...rtl/sv_test/src/sv_test_top.v
              2
 2 F C      dat              /data/PTG/d904686/sv_test/ip_...rtl/sv_test/src/sv_test_top.v
              6
Type number and <Enter> (empty cancels):
```

Appendix III

identifier selection in tags.tcl

- How identifiers are chosen in the script:



```
# Bind all objects to be added to the tag list to the generic callback function
# 'npiCb'.  This call back will add all tags for these objects.
#
::npi_L1::npi_hier_tree_trv_register_cb "npiArrayNet"       "npiCb" "cbList"
::npi_L1::npi_hier_tree_trv_register_cb "npiArrayTypespec"  "npiCb" "cbList"
::npi_L1::npi_hier_tree_trv_register_cb "npiArrayVar"       "npiCb" "cbList"
                    .
                    .
                  [snip]
                    .
                    .
::npi_L1::npi_hier_tree_trv ""
::npi_L1::npi_hier_tree_trv_reset_cb

```

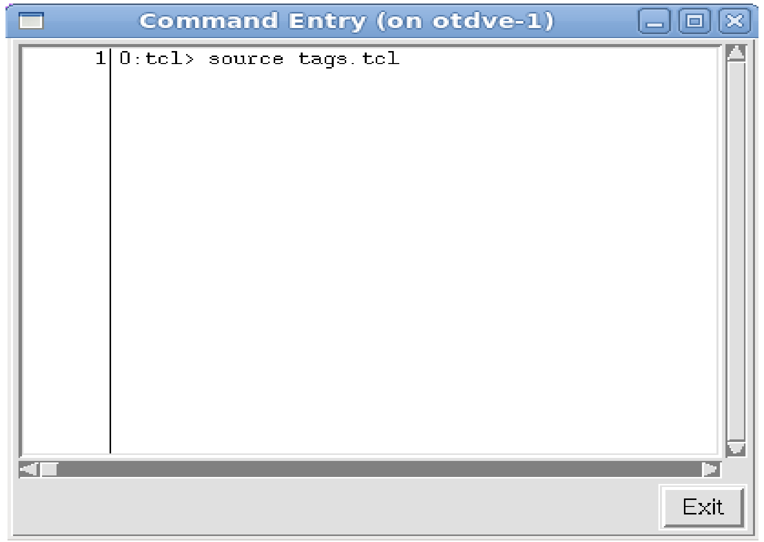
Appendix IV

running the script

- From the command line:

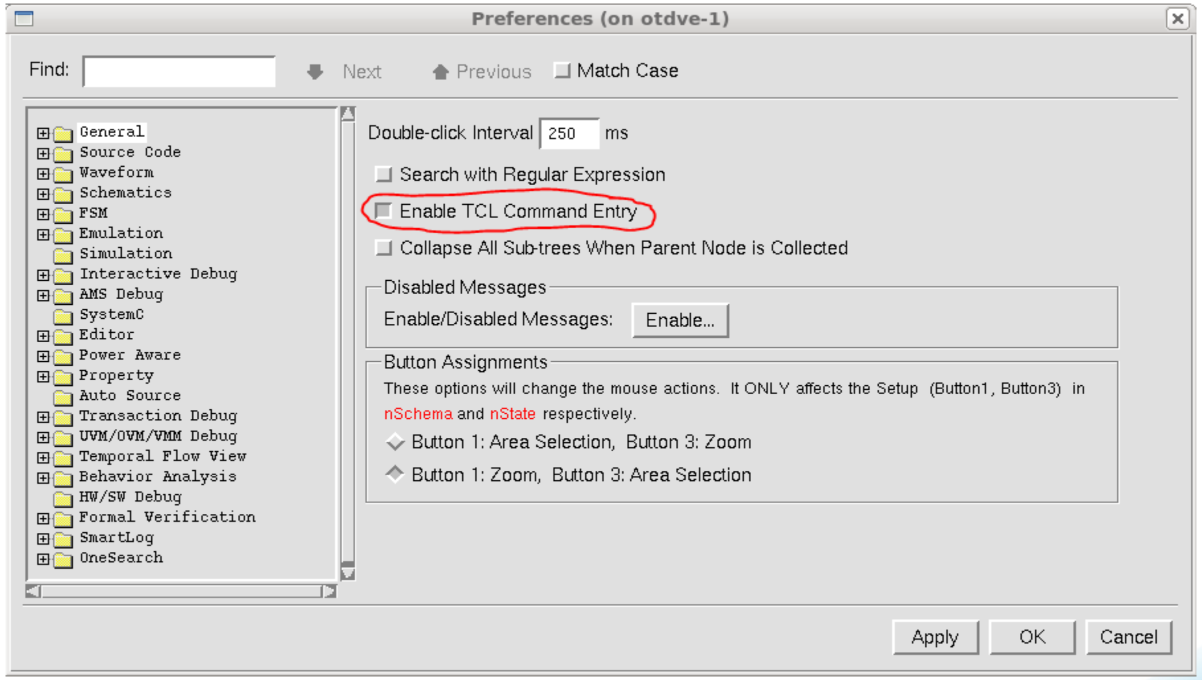
```
verdi –nogui –nologo –q –f design.f –play tags.tcl > /dev/null
```

- or from within Verdi:



Command Entry (on otdve-1)

```
0:tcl> source tags.tcl
```



Preferences (on otdve-1)

Double-click Interval 250 ms

☐ Search with Regular Expression

☐ Enable TCL Command Entry

☐ Collapse All Sub-trees When Parent Node is Collected

Disabled Messages

Enable/Disabled Messages: Enable...

Button Assignments

These options will change the mouse actions. It ONLY affects the Setup (Button1, Button3) in nSchema and nState respectively.

◇ Button 1: Area Selection, Button 3: Zoom

◇ Button 1: Zoom, Button 3: Area Selection