Verilog-Mode

Still Reducing the Veri-Tedium

http://www.veripool.org/papers

/*AUTOAUTHOR*/
Wilson Snyder
wsnyder@wsnyder.org

Last updated Dec, 2016

Verilog Mode
Emacs

Agenda

- The Tedium of Verilog
  - What do I mean by Tedium?
  - Why bother to reduce it?
  - How do we reduce it?
- Verilog-mode Features
  - Wires, Regs, Null Modules, etc…
  - Instantiations
- Help and Support

Verilog Mode
Emacs

Module Tedium? (1995)

```
module tedium (i1,i2,o1,o2);

input  i1,i2;
output o1,o2;

reg    o1;
wire   o2;
wire   inter1;

always @(i1 or i2 or inter1)
  o1 = i1 | i2 | inter1;

sub s1 (.i  (i1),
        .o  (o1),
        .inter1 (inter1));

sub s2 (.i  (i2),
        .o  (o2),
        .inter1 (inter1));

endmodule
```

Argument list is same as input/output statements.

Regs needed for outputs.

Wires needed for interconnections.

Sensitivity lists.

Named based instantiations mostly replicate input/outputs from the sub module.

Verilog Mode
Emacs

Module Tedium? (2009)

```
module tedium (
   input  i1, i2,
   output o1, o2);

reg    o1;
wire   o2;
wire   inter1;

always @*
  o1 = i1 | i2 | inter1;

sub s1 (.i (i1),
        .o (o2),
        .*);

sub s2 (.i (i1),
        .o (o2),
        .*);

endmodule
```

Regs still needed for outputs.
(“output reg” somewhat better)

Wires still needed for interconnections.

@* - Proposed based on Verilog-Mode!

.* - Saves lots of typing,
But can’t see what’s being connected!

When connections aren’t simple, you still need to make connections by hand

Verilog Mode
Emacs

Why eliminate redundancy?

- Reduce spins on fixing lint or compiler warnings
- Reduce sensitivity problems
  - If you forget (or don't have) a linter, these are horrible to debug!
- Make it easier to name signals consistently through the hierarchy
  - Reduce cut & paste errors on multiple instantiations.
  - Make it more obvious what a signal does.
- Reducing the number of lines is goodness alone.
  - Less code to "look" at.
  - Less time typing.

Verilog Mode
Emacs

What would we like in a fix?

- Don’t want a new language
  - All tools would need a upgrade!
  - (SystemVerilog unfortunately faces this hurdle.)
- Don’t want a preprocessor
  - Yet another tool to add to the flow!
  - Would need all users to have the preprocessor!
- Would like code to be completely "valid" Verilog.
  - Want non-tool users to remain happy.
  - Can always edit code without the program.
- Want it trivial to learn basic functions
  - Let the user’s pick up new features as they need them.
- Net result: NO disadvantage to using it
  - Well adopted, including AMCC, ARM, Agere, Altera, AMD, Analog, Cavium, Cisco, Cray, Cypress, Freescale, HP, Infineon, Intel, LSI, MIPS, Maxtor, Micron, NXP, TI, Vitesse, …

Verilog Mode
Emacs

Idea… Use comments!

/*AUTOINST*/ is a metacomment.

The program replaces the text after the comment with the sensitivity list.

```
sub s1 (/*AUTOINST*/);
```

```
sub s1 (/*AUTOINST*/
        .i  (i),
        .o  (o));
```

```
{edit ports of sub}
sub s1 (/*AUTOINST*/
        .i  (i),
        .o  (o));
```

```
sub s1 (/*AUTOINST*/
        .i  (i),
        .new(new),
        .o  (o));
```

If you then edit it, just rerun.

Verilog-Mode for Emacs

- This expansion is best if in the editor
  - You can “see” the expansion and edit as needed
- Verilog package for Emacs
  - Co-Author Michael McNamara `<mac@verilog.com>`
  - Auto highlighting of keywords
  - Standardized indentation
- Reads & expand /*AUTOs*/
  - Magic key sequence for inject/expand/deexpand

C-c C-z: Inject AUTOs

With this key sequence,
Verilog-Mode adds /*AUTOs*/ to old designs!

```
always @(en or a)
  z = a & ena;

submod s (.out  (out),
          .uniq  (u),
          .in   (in));
```

GNU Emacs (Verilog-Mode)

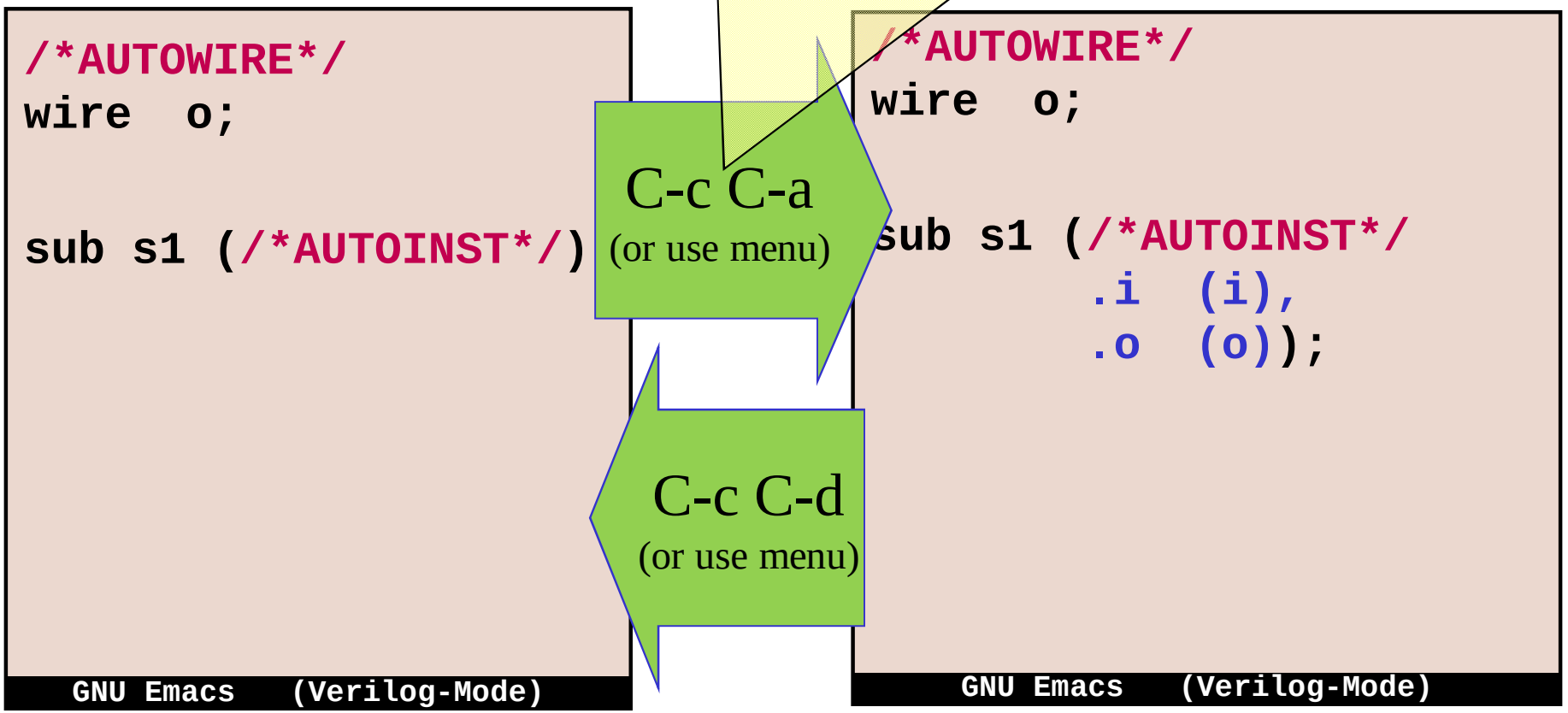
C-c C-z
(or use menu)

```
always @(/*AS*/a or ena)
  z = a & ena;

submod s (.uniq (u),
          /*AUTOINST*/
          // Outputs
          .out  (out),
          // Inputs
          .in   (in));
```

GNU Emacs (Verilog-Mode)

C-c C-a and C-c C-d

With this key sequence,
Verilog-Mode parses the verilog code, and expands the text after any /*AUTO*/ comments.



Verilog Mode
Emacs

But the vi users revolt!

- Call Emacs as part of your flow/other editor
  - `emacs –f verilog-batch-auto {filename.v}`
  - Likewise `verilog-batch-delete-auto`, `verilog-batch-inject-auto`, `verilog-batch-indent`
- Alternatively, the code is always valid!
  - Anyone can simply edit the code and not care about Verilog-Mode.
    - Of course, they need to manually update what would have been done automatically.

Verilog Mode
Emacs

Agenda

- The Tedium of Verilog
  - What do I mean by Tedium?
  - Why bother to reduce it?
  - How do we reduce it?
- Verilog-mode Features
  - Wires, Regs, Null Modules, etc…
  - Instantiations
- Help and Support

Verilog Mode
Emacs

Sensitivity Lists

Alternatively, /*AS*/ is short for /*AUTOSENSE*/

Note “q” is a output, so doesn’t end up in the list.

```
always @ (/*AS*/)
  begin
    if (x) q = a;
    else if (y) q = b;
    else q = c;
  end
```

GNU Emacs (Verilog-Mode)

```
always @ (/*AS*/
          a or b or c
          or x or y)
  begin
    if (x) q = a;
    else if (y) q = b;
    else q = c;
  end
```

GNU Emacs (Verilog-Mode)

Verilog-2001 took this idea from Verilog-Mode and created “`always @*`”

I’d suggest using @* and only use /*AS*/ when you want to see what a large block is sensitive to.

Argument Lists

/*AUTOARG*/ parses the input/output/inout statements.

Or, Verilog-2001 allows ANSI format. Make a team decision which to adopt.

```
module m (/*AUTOARG*/)
  input a;
  input b;
  output [31:0] q;

  …
```

GNU Emacs (Verilog-Mode)

```
module m (/*AUTOARG*/
  // Inputs
  a, b
  // Outputs
  q)

  input a;
  input b;
  output [31:0] q;
```

GNU Emacs (Verilog-Mode)

Automatic Wires

/*AUTOWIRE*/ takes the outputs of sub modules and declares wires for them (if needed -- you can declare them yourself).

```
 …
/*AUTOWIRE*/
/*AUTOREG*/

a a (// Outputs
     .bus   (bus[0]),
     .z     (z));

b b (// Outputs
     .bus   (bus[1]),
     .y     (y));
```

GNU Emacs (Verilog-Mode)

```
/*AUTOWIRE*/
// Beginning of autos
wire [1:0] bus; // From a,b
wire       y;   // From b
wire       z;   // From a
// End of automatics

/*AUTOREG*/

a a (
     // Outputs
     .bus   (bus[0]),
     .z     (z));

b b (
     // Outputs
     .bus   (bus[1]),
     .y     (y));
```

GNU Emacs (Verilog-Mode)

Verilog Mode
Emacs

Automatic Registers

```
…
output [1:0] from_a_reg;
output       not_a_reg;

/*AUTOWIRE*/
/*AUTOREG*/

wire not_a_reg = 1'b1;
```

GNU Emacs (Verilog-Mode)

/*AUTOREG*/ saves having to duplicate reg statements for nets declared as outputs. (If it's declared as a wire, it will be ignored, of course.)

```
output [1:0] from_a_reg;
output       not_a_reg;

 /*AUTOWIRE*/
 /*AUTOREG*/
 // Beginning of autos
 reg [1:0] from_a_reg;
 // End of automatics

 wire not_a_reg = 1'b1;

 always

   … from_a_reg = 2'b00;
```

GNU Emacs (Verilog-Mode)

Verilog Mode
Emacs

Resetting Signals

/*AUTORESET*/ will read signals in the always that don't have a reset, and reset them.

Also works in "always @*" it will use = instead of <=.

```
reg [1:0] a;

always @(posedge clk)
  if (reset) begin
    fsm <= ST_RESET;
    /*AUTORESET*/
  end
  else begin
    a <= b;
    fsm <= ST_OTHER;
  end
end
```

GNU Emacs (Verilog-Mode)

```
reg [1:0] a;

always @(posedge clk)
  if (reset) begin
    fsm <= ST_RESET;
    /*AUTORESET*/
    a <= 2'b0;
  end
  else begin
    a <= b;
    fsm <= ST_OTHER;
  end
```

GNU Emacs (Verilog-Mode)

Verilog Mode
Emacs

State Machines

```
parameter [2:0] // synopsys enum mysm
    SM_IDLE = 3'b000,
    SM_ACT  = 3'b100;

reg [2:0]       // synopsys state_vector mysm
    state_r, state_e1;

/*AUTOASCIIENUM("state_r", "_stateascii_r", "sm_")*/
```

GNU Emacs (Verilog-Mode)

Prefix to remove from ASCII states.

Sized for longest text.

```
/*AUTOASCIIENUM("state_r", "_stateascii_r", "sm_")*/
reg [31:0] _stateascii_r;
always @(state_r)
   casez ({state_r})
      SM_IDLE: _stateascii_r = "idle";
      SM_ACT:  _stateascii_r = "act ";
      default: _stateascii_r = "%Err";
   endcase
```

GNU Emacs (Verilog-Mode)

Null/Stub Modules, Tieoffs

AUTOINOUTMODULE will copy I/O from another module. AUTOTIEOFF will terminate undriven outputs, and AUTOUNUSED will terminate unused inputs.

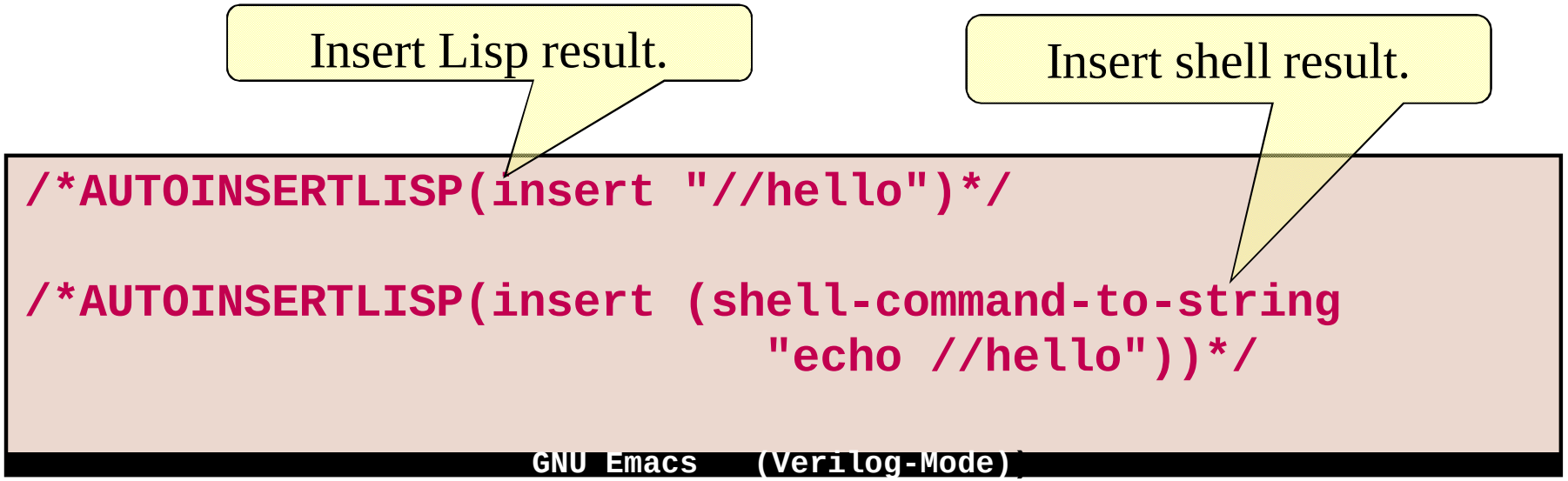
```
module ModStub (
      /*AUTOINOUTMODULE
                  ("Mod")*/
      );

  /*AUTOWIRE*/
  /*AUTOREG*/
  /*AUTOTIEOFF*/

  wire _unused_ok = &{
      /*AUTOUNUSED*/
      1'b0};

endmodule
```

GNU Emacs (Verilog-Mode)

```
module ModStub (
      /*AUTOINOUTMODULE
                  ("Mod")*/
      input         mod_in,
      output [2:0] mod_out
      );

  /*AUTOWIRE*/
  /*AUTOREG*/
  /*AUTOTIEOFF*/
  wire [2:0] mod_out = 3'b0;

  wire _unused_ok = &{
      /*AUTOUNUSED*/
      mod_in,
      1'b0};

endmodule
```

GNU Emacs (Verilog-Mode)

Script Insertions




```
/*AUTOINSERTLISP(insert "//hello")*/
//hello

/*AUTOINSERTLISP(insert (shell-command-to-string
                       "echo //hello"))*/
//hello
```

GNU Emacs (Verilog-Mode)

`ifdefs

We manually put in the ifdef, as we would have if not using Verilog-mode.

Verilog-mode a signal referenced before the AUTOARG, leaves that text alone, and omits that signal in its output.

```
module m (
`ifdef c_input
  c,
`endif
  /*AUTOARG*/)

  input a;
  input b;

`ifdef c_input
  input c;
`endif
```

GNU Emacs (Verilog-Mode)

```
module m (
`ifdef c_input
  c,
`endif
  /*AUTOARG*/
  // Inputs
  a, b)

  input a;
  input b;

`ifdef c_input
  input c;
`endif
```

GNU Emacs (Verilog-Mo

Why not automatic?

Obviously, the `ifdefs would have to be put into the output text (for it to work for both the defined & undefined cases.)

One ifdef would work, but consider multiple nested ifdefs each on overlapping signals. The algorithm gets horribly complex for the other commands (AUTOWIRE).

Verilog Mode
Emacs

Agenda

- The Tedium of Verilog
  - What do I mean by Tedium?
  - Why bother to reduce it?
  - How do we reduce it?
- Verilog-mode Features
  - Wires, Regs, Null Modules, etc…
  - Instantiations
- Help and Support

Verilog Mode
Emacs

Simple Instantiations

/*AUTOINST*/
Look for the submod.v file, read its in/outputs.

```
submod s (/*AUTOINST*/);
```

```
module submod;
  output out;
  input in;

  …

endmodule
```

GNU Emacs (Verilog-Mode)

```
submod s (/*AUTOINST*/
          // Outputs
          .out  (out),
          // Inputs
          .in   (in));
```

Keep signal names consistent!

Note the simplest and most obvious case is to have the signal name on the upper level of hierarchy match the name on the lower level. Try to do this when possible.

Occasionally two designers will interconnect designs with different names. Rather then just connecting them up, it's a 30 second job to use *vrename* from my Verilog-Perl suite to make them consistent.

Verilog Mode
Emacs

Instantiation Example

```
module pci_mas
       (/*AUTOARG*/
       trdy);
  input  trdy;

  …
```

```
module pci_tgt
       (/*AUTOARG*/
       irdy);
  input  irdy;

  …
```

```
module pci (/*AUTOARG*/
            irdy, trdy);
 input irdy;
 input trdy;
 /*AUTOWIRE*/
 // Beginning of autos
 // End of automatics


 pci_mas mas (/*AUTOINST*/
        // Inputs
        .trdy     (trdy));


 pci_tgt tgt (/*AUTOINST*/
        // Inputs
        .irdy     (irdy));
```

Verilog Mode
Emacs

Instantiation Example

```
module pci_mas
       (/*AUTOARG*/
       trdy, mas_busy);
  input  trdy;
  output mas_busy;
  …
```

```
module pci_tgt
       (/*AUTOARG*/
       irdy, mas_busy);
  input  irdy;
  input  mas_busy;
  …
```

```
module pci (/*AUTOARG*/
            irdy, trdy);
 input irdy;
 input trdy;
 /*AUTOWIRE*/
 // Beginning of autos
 wire mas_busy;   // From mas.v
 // End of automatics

 pci_mas mas (/*AUTOINST*/
        // Outputs
        .mas_busy (mas_busy),
        // Inputs
        .trdy     (trdy));
 pci_tgt tgt (/*AUTOINST*/
        // Inputs
        .irdy     (irdy),
        .mas_busy (mas_busy));
```

Verilog Mode
Emacs

Exceptions to Instantiations

Method 1: AUTO_TEMPLATE lists exceptions for "submod." The ports need not exist.
(This is better if submod occurs many times.)

```
/* submod AUTO_TEMPLATE (
   .z (otherz),
   );
*/

submod s (// Inputs
          .a (except1),
          /*AUTOINST*/);
```

Method 2: List the signal before the AUTOINST. First put a // Input or // Output comment for AUTOWIRE.

Initial Technique

First time you're instantiating a module, let AUTOINST expand everything. Then cut the lines it inserted out, and edit them to become the template or exceptions.

```
/* submod AUTO_TEMPLATE (
   .z (otherz),
   );
*/

submod s (// Inputs
          .a  (except1),
          /*AUTOINST*/
          .z  (otherz),
          .b  (b));
GNU Emacs    (Verilog-Mode)
```

Signals not mentioned otherwise are direct connects.

Multiple Instantiations

@ in the template takes the leading digits from the reference. (Or next slide.)

```
/* submod AUTO_TEMPLATE (
    .z (out[@]),
    .a (invec@[]));
*/

submod i0 (/*AUTOINST*/);

submod i1 (/*AUTOINST*/);

submod i2 (/*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)

[] takes the bit range for the bus from the referenced module. Generally, always just add [].

```
/* submod AUTO_TEMPLATE (
    .z (out[@]),
    .a (invec@[]));
*/

submod i0 (/*AUTOINST*/
           .z (out[0]),
           .a (invec0[31:0]));

submod i1 (/*AUTOINST*/
           .z (out[1]),
           .a (invec1[31:0]));
```

GNU Emacs (Verilog-Mode)

Overriding @

A regexp after AUTO_TEMPLATE specifies what to use for @ instead of last digits in cell name.
Below, @ will get name of module.

```
/* submod AUTO_TEMPLATE
             "\(.*\)" (
   .z (out_@[]));
*/

submod i0 (/*AUTOINST*/);

submod i1 (/*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)

```
/* submod AUTO_TEMPLATE (
             "\(.*\)" (
   .z (out_@[]));
*/

submod i0 (/*AUTOINST*/
           .z (out_i0));

submod i1 (/*AUTOINST*/
           .z (out_i1));
```

GNU Emacs (Verilog-Mode)

Verilog Mode
Emacs

Instantiations using LISP

@"{lisp_expression}"
Decodes in this case to:
in[31-{the_instant_number}]

```
/* buffer AUTO_TEMPLATE (
   .z (out[@]),
   .a (in[@"(- 31 @)"]));
*/

buffer i0 (/*AUTOINST*/);

buffer i1 (/*AUTOINST*/);

buffer i2 (/*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)

Predefined Variables

See the documentation for variables that are useful in Lisp templates:
vl-cell-type, vl-cell-name, vl-modport, vl-name, vl-width, vl-dir.

```
/* buffer AUTO_TEMPLATE (
   .z (out[@]),
   .a (in[@"(- 31 @)"]));
*/

buffer i0 (/*AUTOINST*/
           .z  (out[0]),
           .a  (in[31]));

buffer i1 (/*AUTOINST*/
           .z  (out[1]),
           .a  (in[30]));
```

GNU Emacs (Verilog-Mode)

Instantiations using RegExps

.\(\) indicates a Emacs regular expression.

@ indicates “match-a-number” Shorthand for \([0-9]+\)

```
/* submod AUTO_TEMPLATE (
   .\(.*[^0-9]\)@   (\1[\2]),
   );*/

submod i (/*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)

```
/* submod AUTO_TEMPLATE (
   .\(.*[^0-9]\)@   (\1[\2]),
   );*/

submod i (/*AUTOINST*/
          .vec2    (vec[2]),
          .vec1    (vec[1]),
          .vec0    (vec[0]),
          .scalar  (scalar));
```

GNU Emacs (Verilog-Mode)

Signal name is first \(\) match, substituted for \1.

Bit number is second \(\) match (part of @), substituted for \2.

Instantiations with Parameters

AUTOINSTPARAM is just like AUTOINST, but "connects" parameters.

```
submod #(/*AUTOINSTPARAM*/)
    i (/*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)

```
submod #(/*AUTOINSTPARAM*/
         .WIDTH(WIDTH))
    i (/*AUTOINST*/
       .out   (out));
```

GNU Emacs (Verilog-Mode)

Parameter Values

Often, you want parameters to be "constant" in the parent module. verilog-auto-inst-param-value controls this.

```
submod #(.WIDTH(8))
  i (/*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)

or

```
submod #(.WIDTH(8))
  i (/*AUTOINST*/
    .out(out[WIDTH-1:0]));

// Local Variables:
// verilog-auto-inst-param-value:nil
// End:
```

GNU Emacs (Verilog-Mode)

```
submod #(.WIDTH(8))
  i (/*AUTOINST*/
    .out(out[7:0]));

// Local Variables:
// verilog-auto-inst-param-value:t
// End:
```

GNU Emacs (Verilog-Mode)

Exclude AUTOOUTPUT

Techniques to exclude signals from AUTOOUTPUT (or AUTOINPUT etc).

1. Declare “fake” output

```
`ifdef NEVER
  output out;
`endif

submod i (// Output
          .out(out));
```

GNU Emacs (Verilog-Mode)

2. Use inclusive regexp

```
// Regexp to include
/*AUTOOUTPUT(“in”)*/

submod i (// Output
          .out(out));
```

GNU Emacs (Verilog-Mode)

3. Set the output ignore regexp

```
/*AUTO_LISP
   (setq verilog-auto
         -output-ignore-regexp
     (verilog-regexp-words `(
          “out“
           )))*/

submod i (// Output
          .out(out));
```

GNU Emacs (Verilog-Mode)

4. Use concats to indicate exclusion

```
submod i (// Output
          .out({out}));

// Local Variables:
// verilog-auto-ignore-concat:t
// End:
```

GNU Emacs (Verilog-Mode)

SystemVerilog .*

SystemVerilog .* expands just like AUTOINST.

```
submod i (.*);
```

GNU Emacs (Verilog-Mode)

C-c C-a
(autos)

```
submod i (.*
         .out (out));
```

BUT, on save reverts to .* (unless verilog-auto-save-star set)

C-x C-s
(save)

```
submod i (.*);
```

GNU Emacs (Verilog-Mode)

See Cliff Cumming's Paper

http://www.sunburst-design.com/papers/CummingsSNUG2007Boston_DotStarPorts.pdf

Verilog Mode
Emacs

Where to Find Modules

1. Old Scheme

```
// Local Variables:
// verilog-library-directories:("." "dir1" "dir2" ...)
// End:
```

GNU Emacs (Verilog-Mode)

Jumping: C-c C-d

C-c C-d jumps to the definition of the entered module

2. Better

```
// Local Variables:
// verilog-library-flags:("-y dir1 -y dir2")
// End:
```

GNU Emacs (Verilog-Mode)

3. Best

```
// Local Variables:
// verilog-library-flags:("-f ../../input.vc")
// End:
```

GNU Emacs (Verilog-Mode)

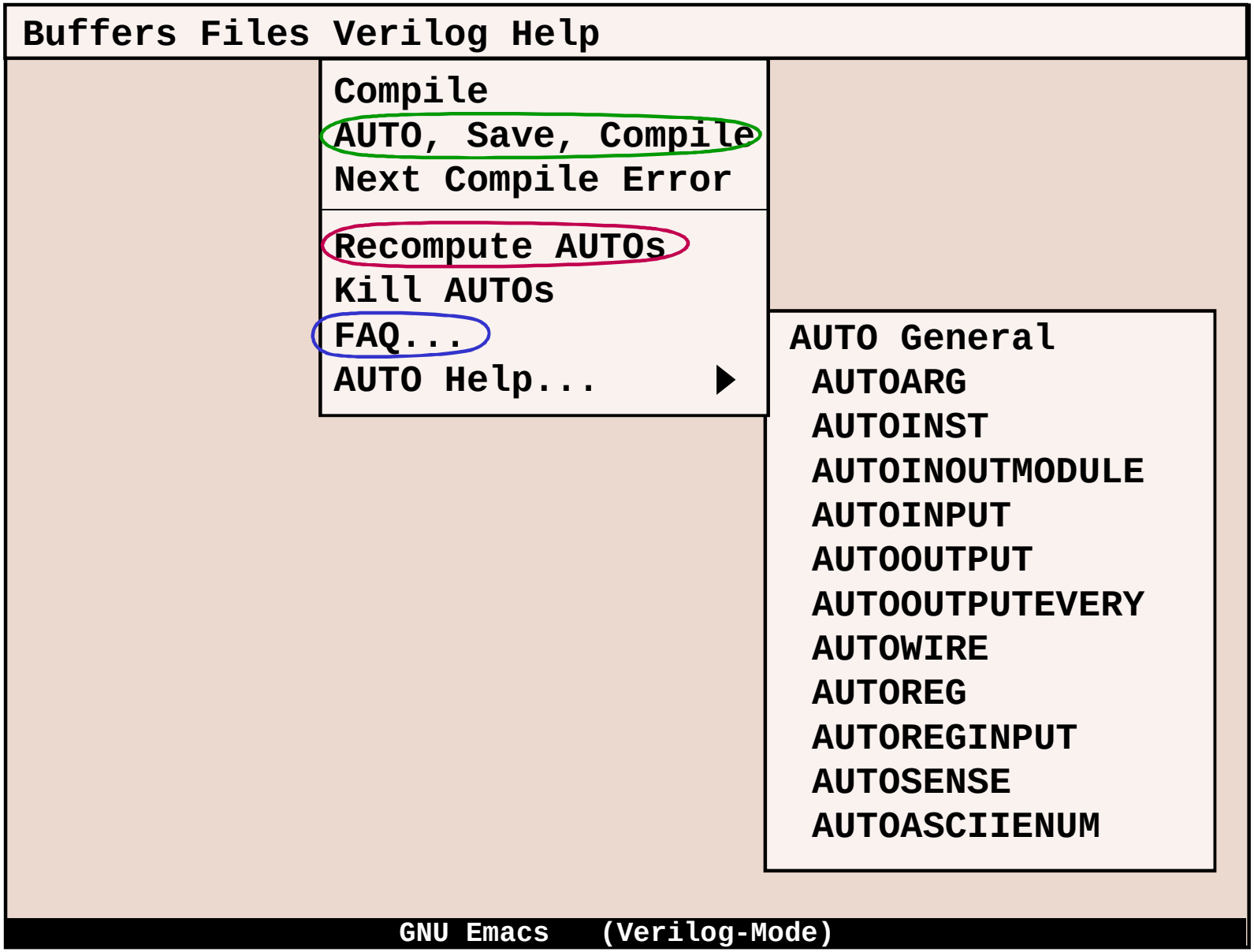
```
// input.vc
-y dir1
-y dir2
```

Best, as input.vc can (should) be the same file you feed your lint/synth/simulator.

Agenda

- The Tedium of Verilog
  - What do I mean by Tedium?
  - Why bother to reduce it?
  - How do we reduce it?
- Verilog-mode Features
  - Wires, Regs, Null Modules, etc…
  - Instantiations
- Help and Support

Verilog Menu Help



Verilog Mode
Emacs

FAQ on Indentations, etc

- Have TAB insert vs. indent:

```
(setq verilog-auto-newline nil)
(setq verilog-tab-always-indent nil)
```

- Lineup signals around “=”:

See `verilog-auto-lineup`

- Don’t add “end // block-name” comments:

```
(setq verilog-auto-endcomments nil)
```

Verilog Mode
Emacs

SystemVerilog Support

- Keyword Highlighting – Full support
- Indentation – Most everything we've seen
- AUTOs – Getting there
  - Logic, bit
  - Typedefs (setq verilog-typedef-regexp "_t$")
  - Multidimensional ports
  - Interfaces as ports
  - .*, .name
- Please let us know what's missing

Homework Assignment

- Homework
  - Due Next week:
    - Install Verilog-Mode
    - Try Inject-Autos
    - Use AUTOINST in one module
  - Grow from there!

Open Source

- The open source design tools are available at http://www.veripool.org
  - These slides + paper at http://www.veripool.org/papers/
  - Bug Reporting there (Please!)

- Additional Tools
  - Schedule::Load – Load Balancing (ala LSF)
  - SystemPerl – /*AUTOs*/ for SystemC
  - Verilog-Mode for Emacs – /*AUTO…*/ Expansion
  - Verilog-Perl – Toolkit with Preprocessing, Renaming, etc
  - Verilator – Compile SystemVerilog into SystemC
  - Vregs – Extract register and class declarations from documentation