Verilog-Mode AUTOS

Reducing the SystemVerilog Tedium

https://www.veripool.org/papers

Wilson Snyder

April 6, 2020

(Updated April 15, 2020)

Verilog Mode Emacs

Agenda

- The Tedium of SystemVerilog
  - 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

Tedium?

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

logic  o1;
wire   o2;
wire   inter1;

always_comb
  o1 = i1 | i2 | inter1;

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

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

endmodule
```

Wires declared for interconnections

BTW, IEEE 2001's "always @*" based on Verilog-Mode!

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

Hand-coded non-direct connections

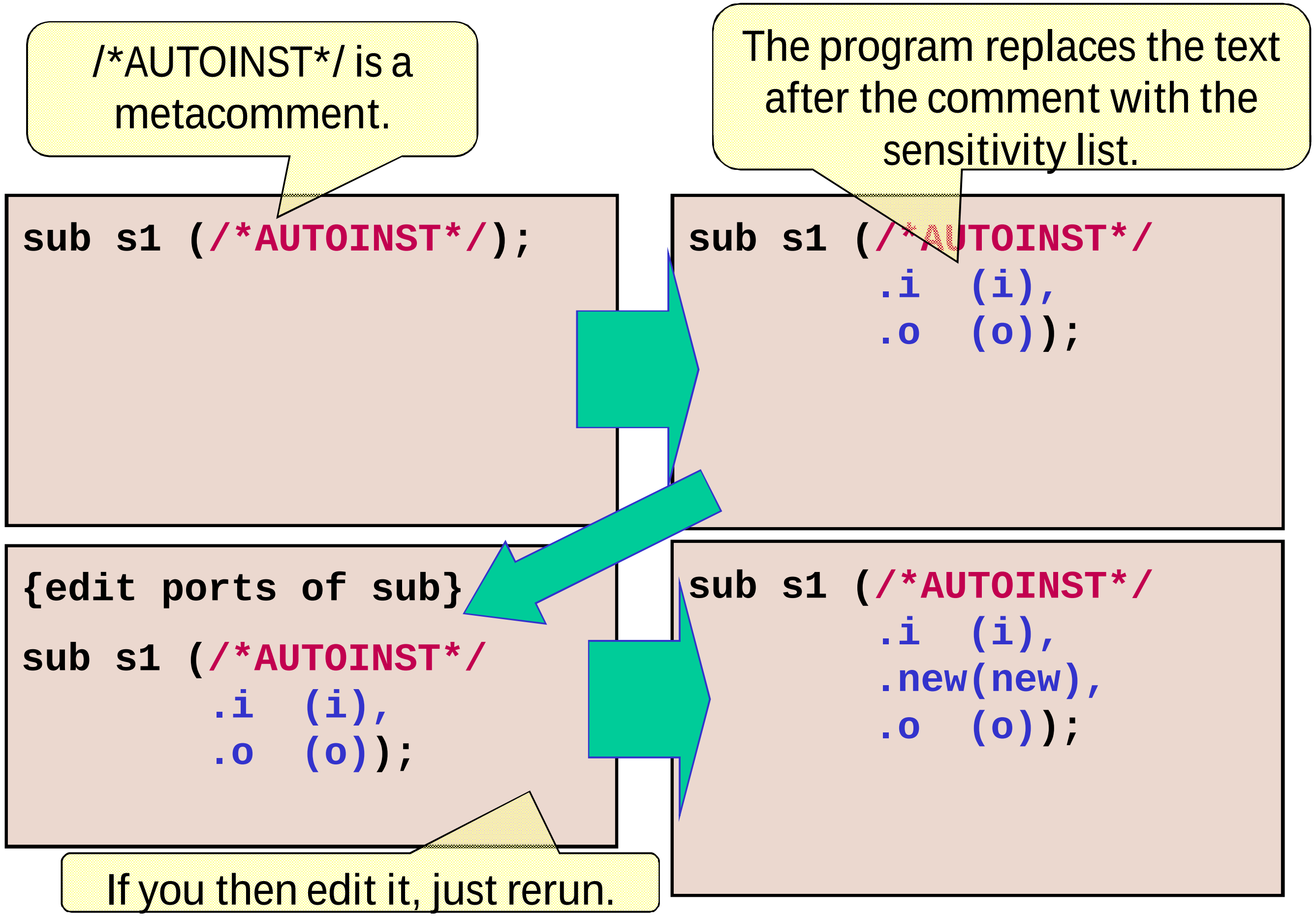
Why eliminate redundancy?

- Faster to modify the code
- Reduces spins on fixing lint or compiler warnings
- 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
  - Less code to "look" at.
  - Less time typing.

What would we like in a fix?

- Don’t want a new language
  - All tools would need a upgrade!
- Don’t want a preprocessor
  - Yet another tool to add to the flow!
  - Would need all users to have the preprocessor!
- Want code always as "valid" SystemVerilog
  - Want non-tool users to remain happy
  - Enable editing code without the tool
- Want trivial to learn basic functions
  - Let the user’s pick up new features as they need them
- Want wide industry use at Arm, Altera, AMD, Analog, Broadcom, Cisco, Cray, Intel, MIPS, Marvell, Qualcomm, TI…
- **Verilog-Mode delivers**

Idea… Use comments!



/*AUTOINST*/ is a metacomment.

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

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

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

```
{edit ports of sub}

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

If you then edit it, just rerun.

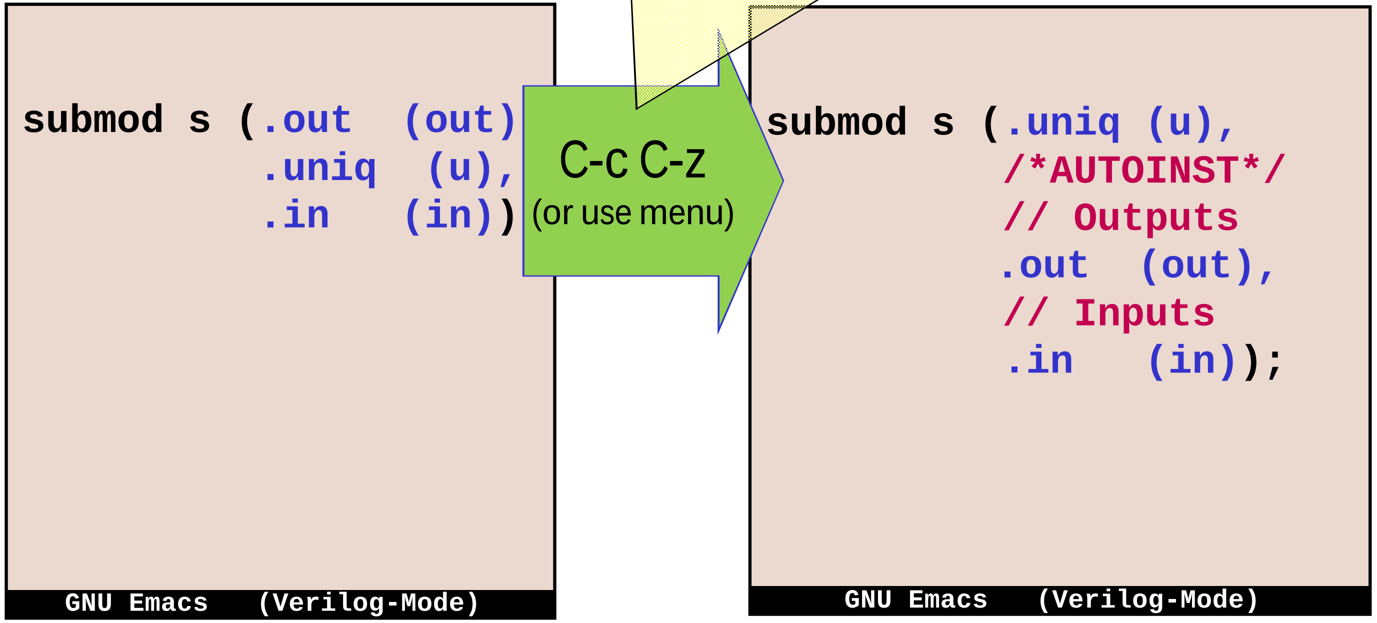
```
sub s1 (/*AUTOINST*/
        .i   (i),
        .new(new),
        .o   (o));
```

Verilog-Mode

- Expansion is best if in the editor
  - “See” the expansion and edit as needed
- Verilog-mode package for Emacs
- Reads & expand /*AUTOs*/
  - Magic key sequence for inject/expand/deexpand
- May use as stand-alone tool, or called from other editors

C-c C-z: Inject AUTOs

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

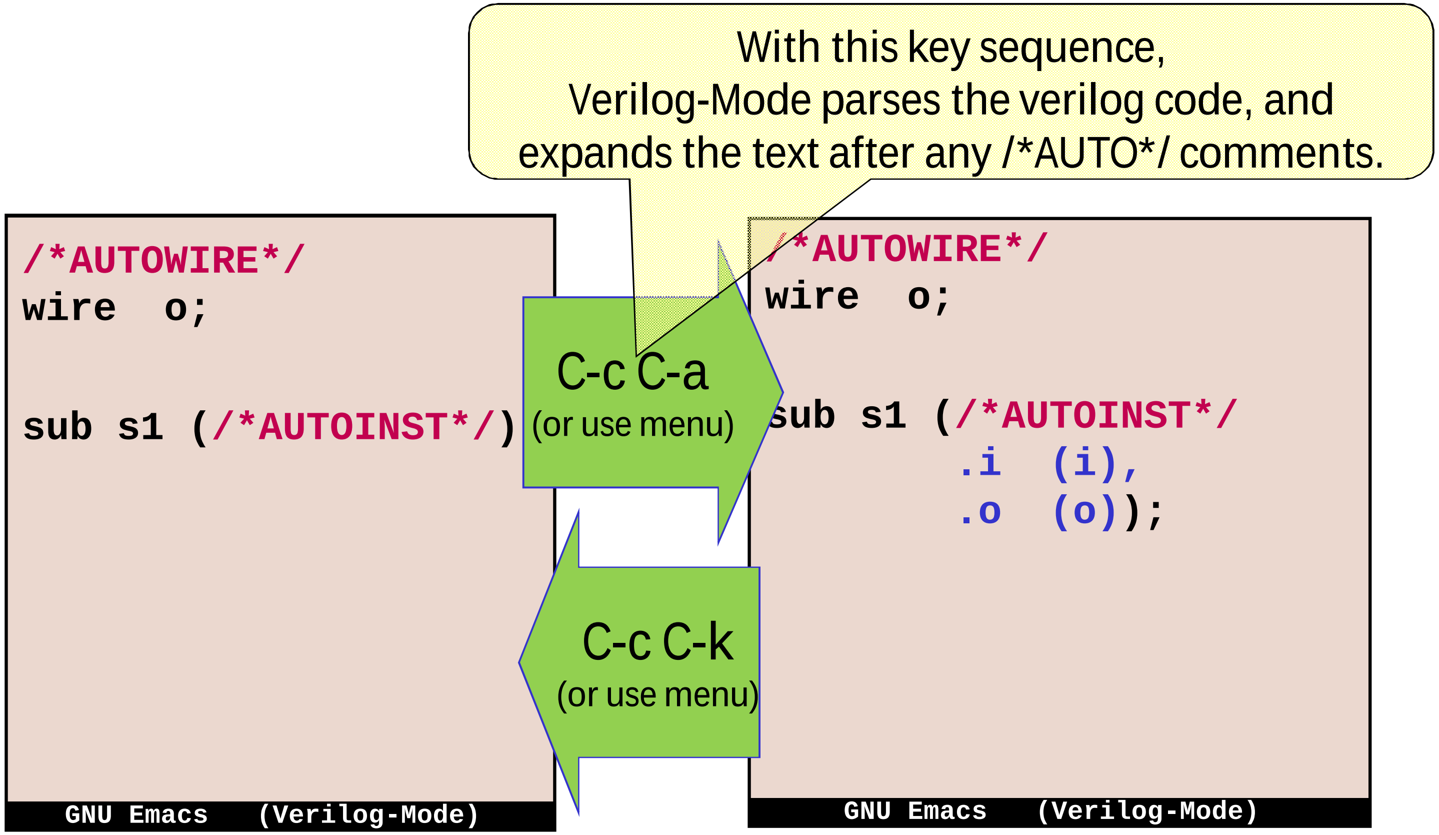


```
submod s (.out   (out)
          .uniq  (u),
          .in    (in))
```

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

or "`$ emacs --batch` *`file.sv`* `-f verilog-batch-auto-inject`"

C-c C-a and C-c C-k



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

```
/*AUTOWIRE*/
wire  o;


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

```
/*AUTOWIRE*/
wire  o;


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

or "`$ emacs --batch file.sv -f verilog-batch-auto`"

But the vi & Eclipse users revolt!

- Call Emacs as part of your flow or other editor
  - `emacs --batch` *`filename.sv`* `–f verilog-batch-auto`
  - Likewise `verilog-batch-delete-auto`,
    `verilog-batch-inject-auto`,
    `verilog-batch-indent`

- Again, 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
  - Thus AUTOs are popular in commercial IP

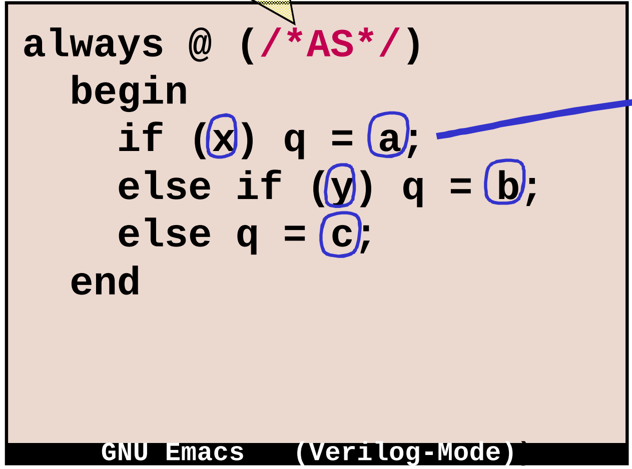
Agenda

- The Tedium of SystemVerilog
  - 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

Sensitivity Lists

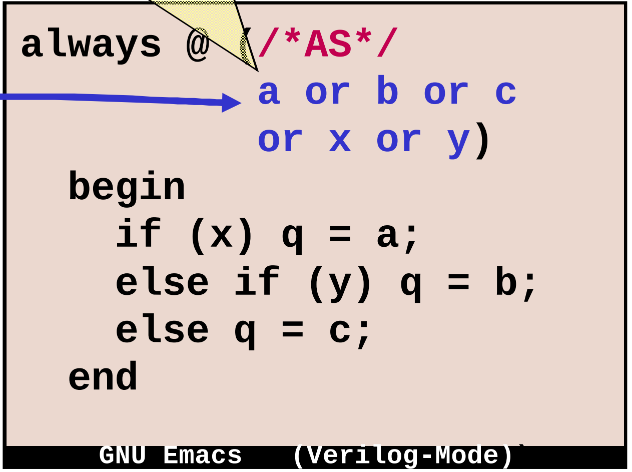
Alternatively, /*AS*/ is short for /*AUTOSENSE*/

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
```

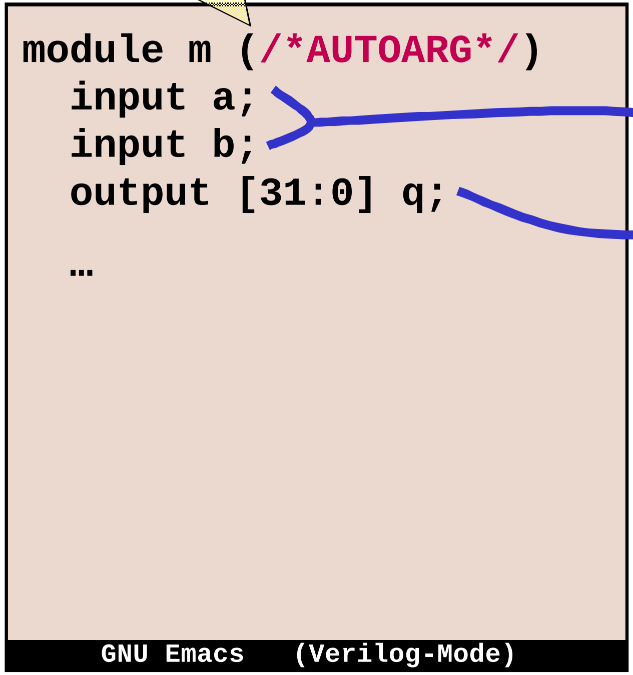
```
 GNU Emacs    (Verilog-Mode)
```

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

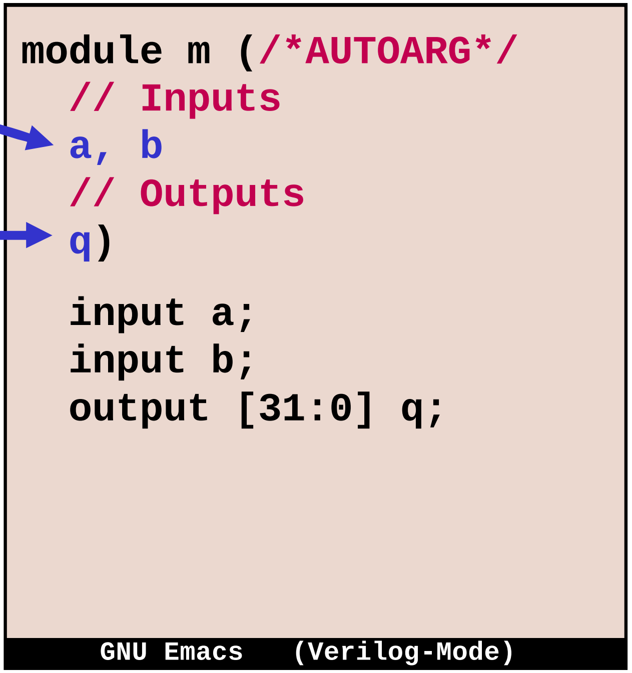
/*AUTOARG*/ parses the input/output/inout statements.

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)

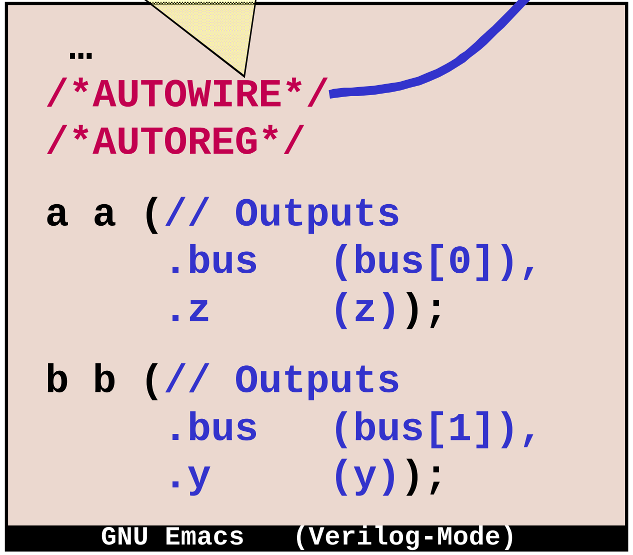


```
module m (/*AUTOARG*/
   // Inputs
   a, b
   // Outputs
   q)

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

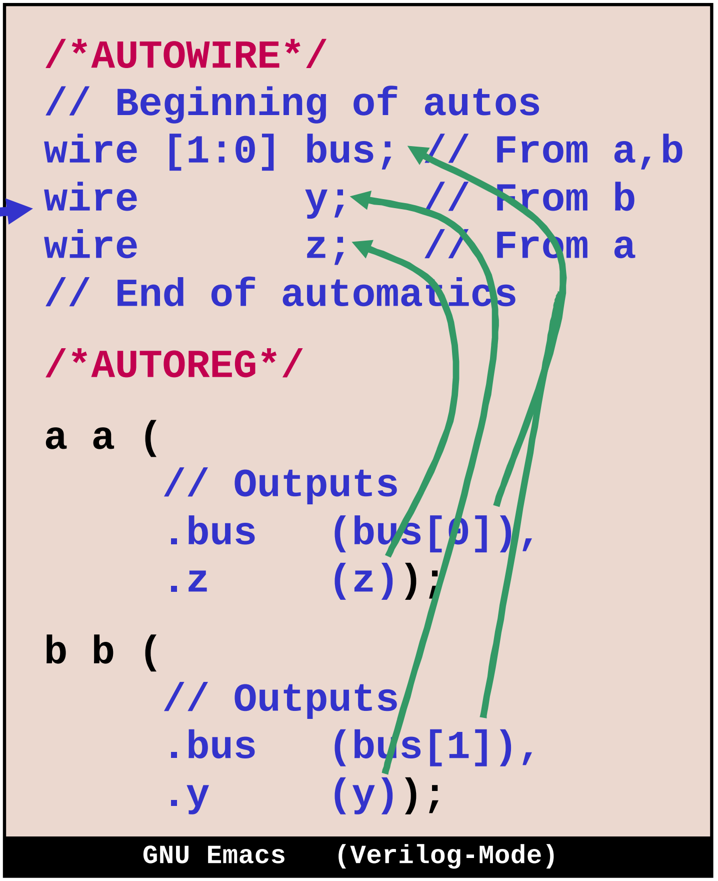
GNU Emacs   (Verilog-Mode)

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));
```

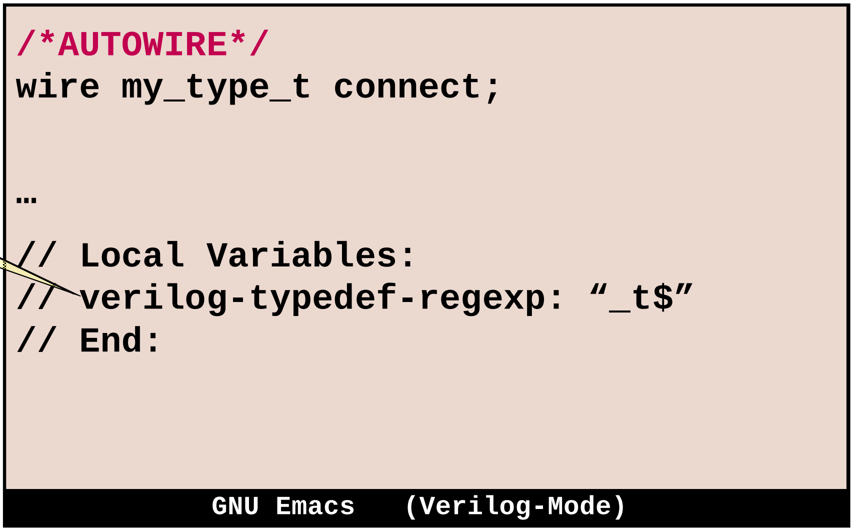


```
/*AUTOWIRE*/
// Beginning of autos
wire [1:0] bus; // From a,b
wire       y;   // From b
wire       z;   // From a
// End of automatics

/*AUTOREG*/

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

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

Verilog Mode Emacs

Datatypes

Verilog-Mode needs a regexp to identify what is a data type



```
/*AUTOWIRE*/
wire my_type_t connect;


…

// Local Variables:
// verilog-typedef-regexp: “_t$”
// End:
```

GNU Emacs   (Verilog-Mode)

Automatic Registers

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

/*AUTOWIRE*/
/*AUTOLOGIC*/

wire not_a_reg = 1'b1;
```

GNU Emacs   (Verilog-Mode)

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

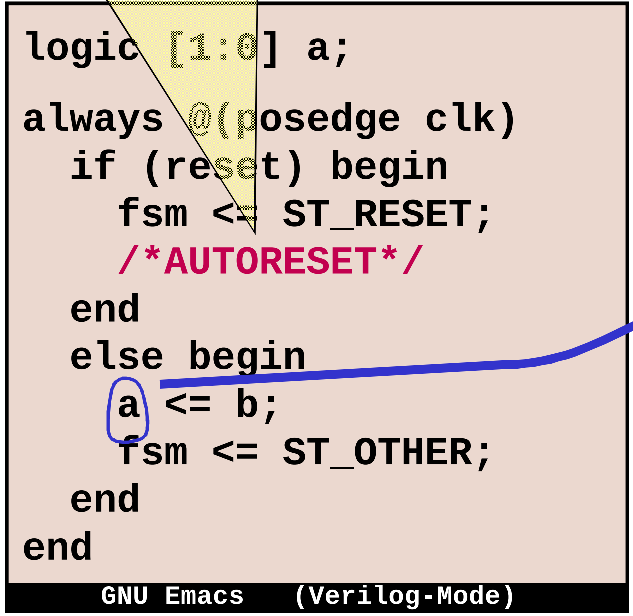
```
output [1:0] from_a_reg;
output       not_a_reg;

 /*AUTOWIRE*/
 /*AUTOLOGIC*/
 // Beginning of autos
 logic [1:0] from_a_reg;
 // End of automatics

 wire not_a_reg = 1'b1;

 always

   … from_a_reg = 2'b00;
```

GNU Emacs   (Verilog-Mode)

Resetting Signals

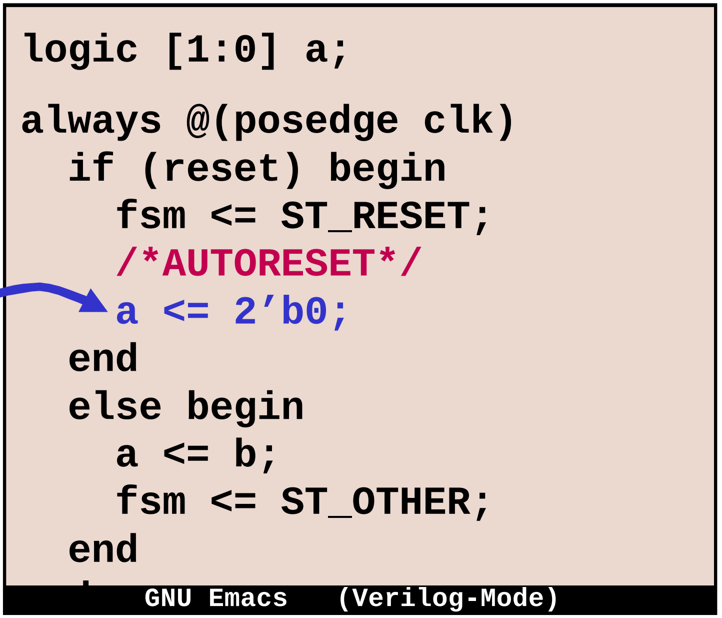
/*AUTORESET*/ will read signals in the always that don't have a reset, and reset them.

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



```
logic [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)

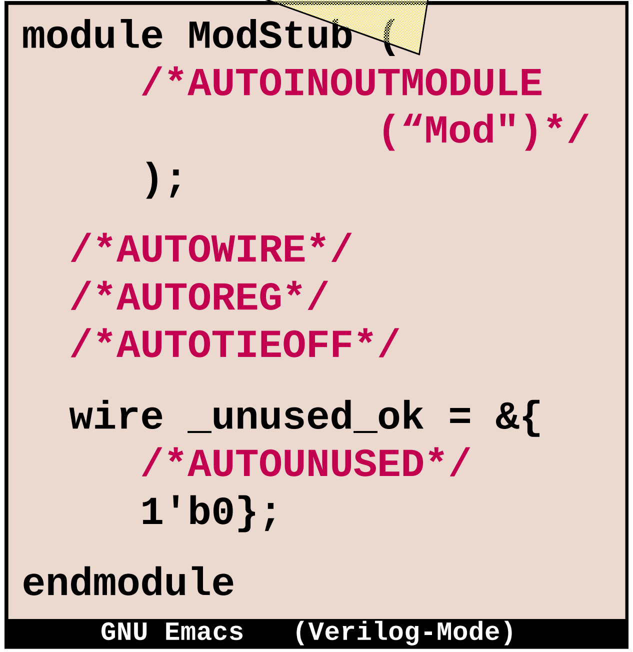


```
logic [1:0] a;

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)

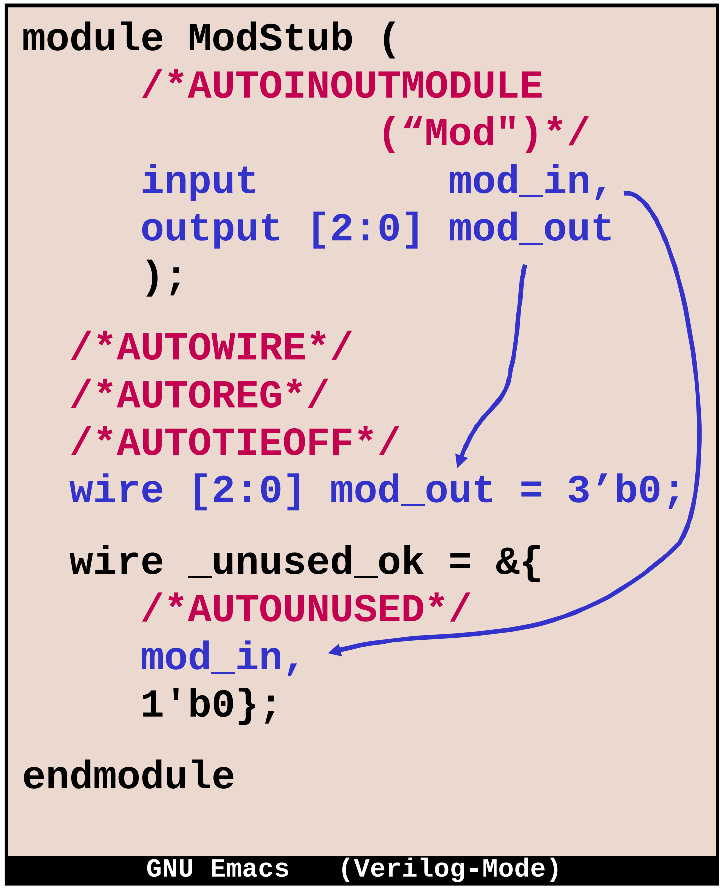
Null/Stub Modules, Tieoffs

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
```

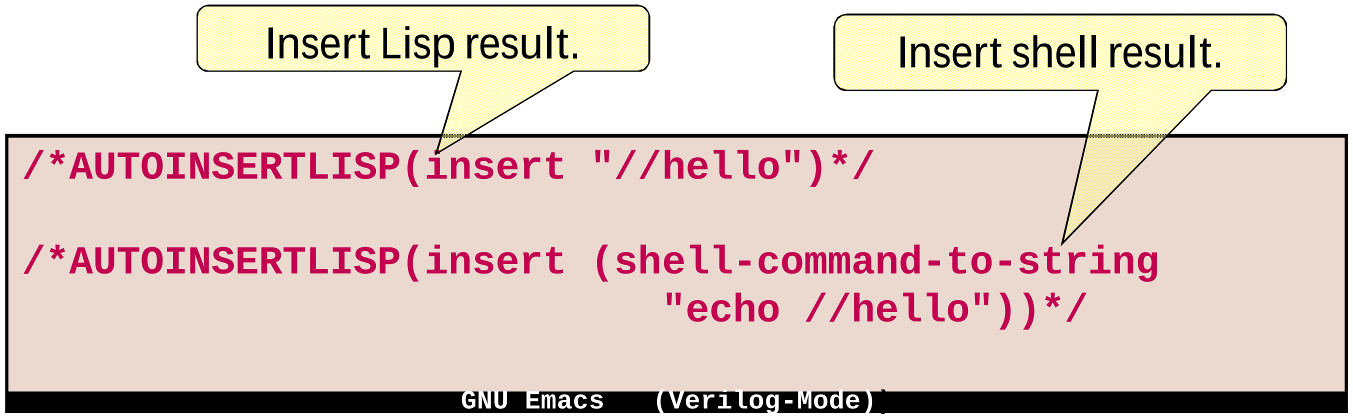
GNU Emacs   (Verilog-Mode)



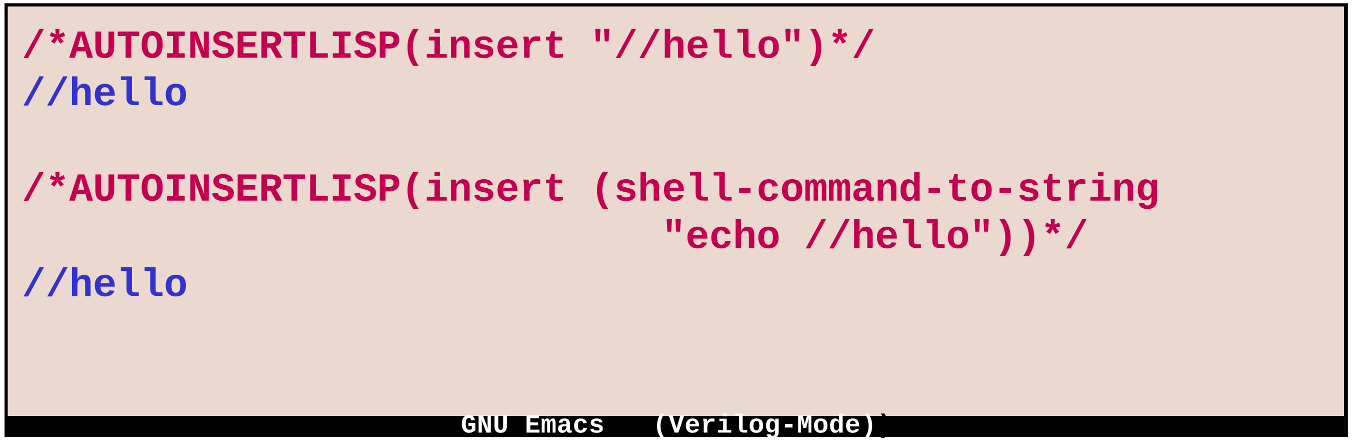
```
module ModStub (
     /*AUTOINOUTMODULE
               (“Mod")*/
     input        mod_in,
     output [2:0] mod_out
     );

   /*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")*/

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

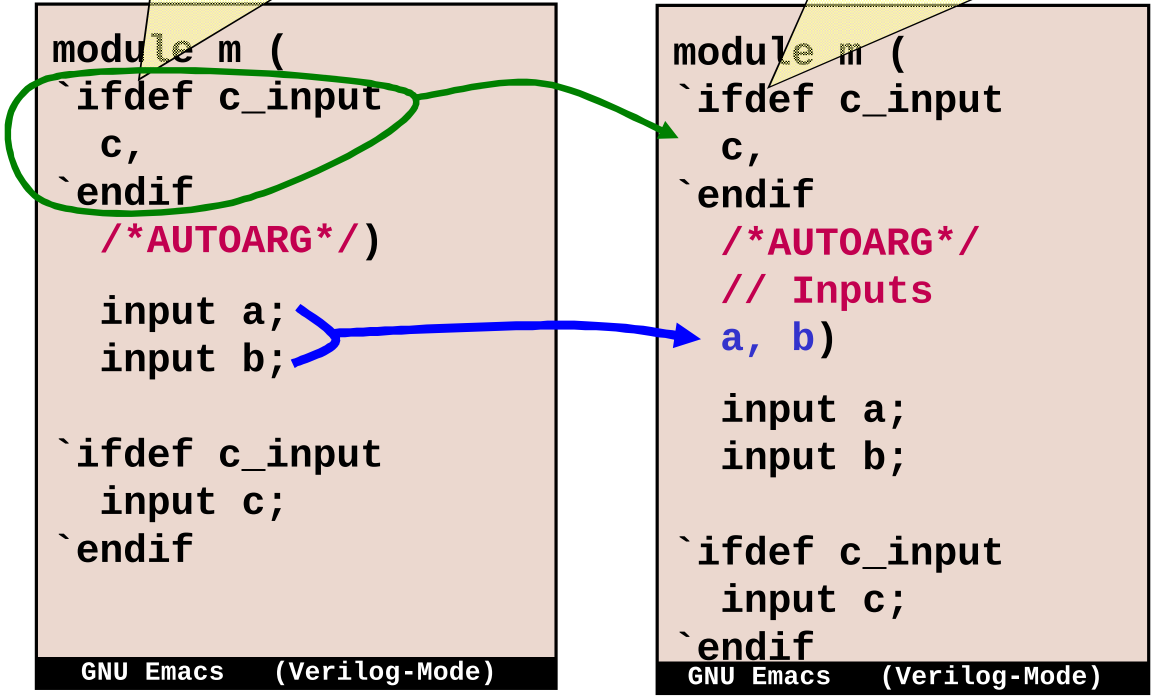


```
/*AUTOINSERTLISP(insert "//hello")*/
//hello

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

`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-Mode)
```

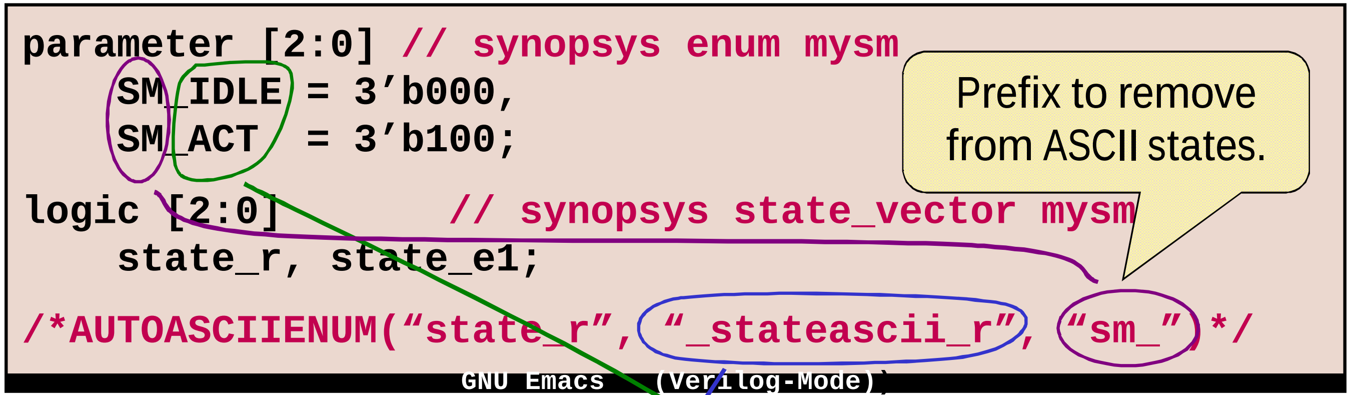
Why not automatic?

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 AUTOWIRE etc.

State Machines

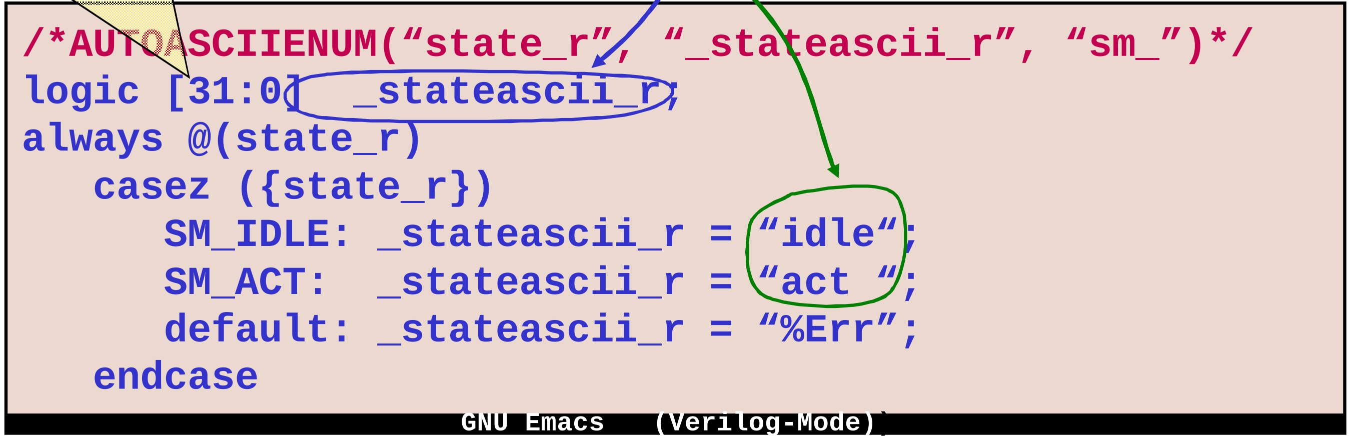
wsnyder



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

logic [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.



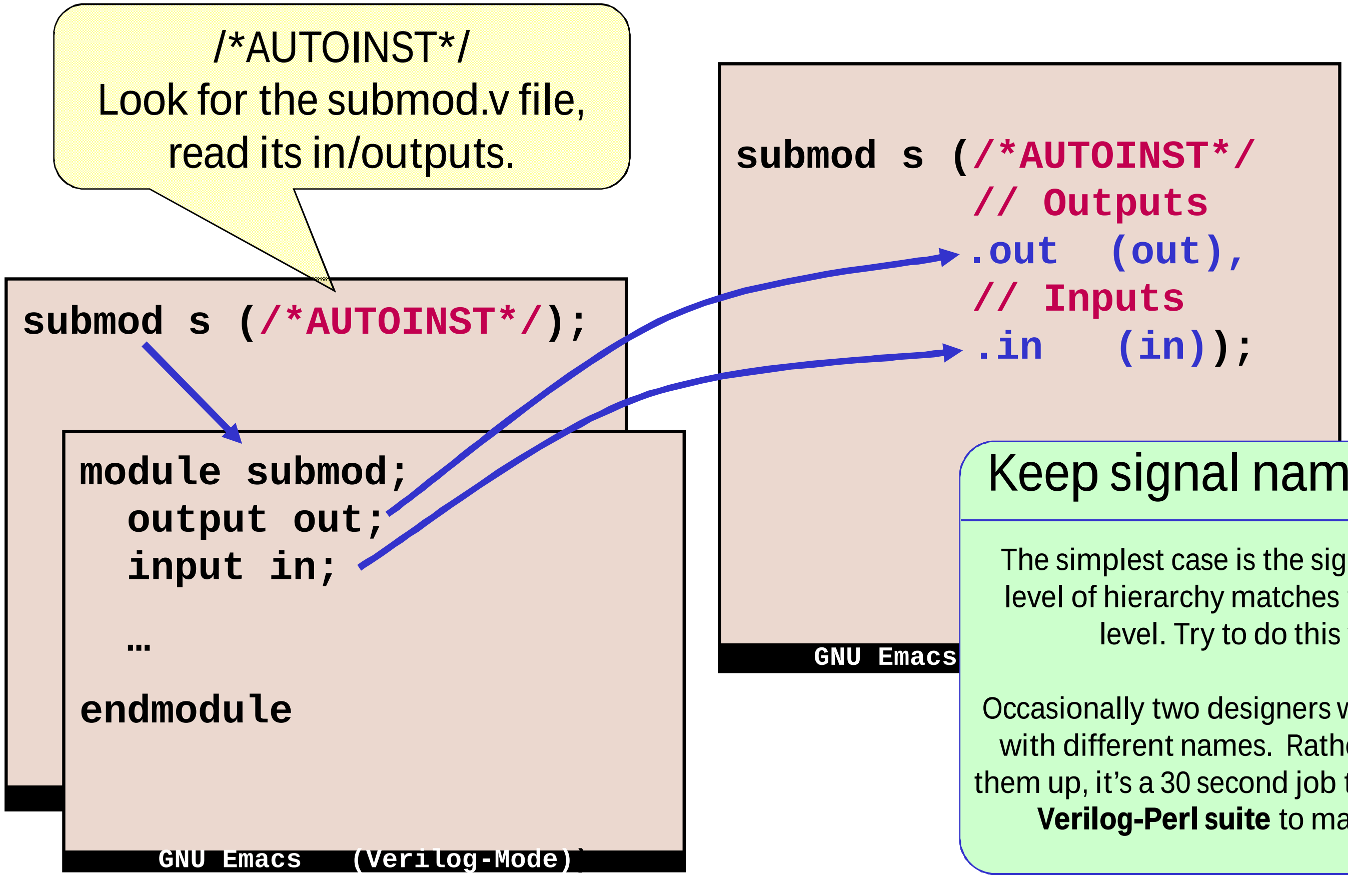
```
/*AUTOASCIIENUM("state_r", "_stateascii_r", "sm_")*/
logic [31:0]   _stateascii_r;
always @(state_r)
   casez ({state_r})
      SM_IDLE: _stateascii_r = "idle";
      SM_ACT:  _stateascii_r = "act ";
      default: _stateascii_r = "%Err";
   endcase
                GNU Emacs    (Verilog-Mode)
```

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
```

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

Keep signal names consistent!

The simplest case is the signal name on the upper level of hierarchy matches 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.

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));
```

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

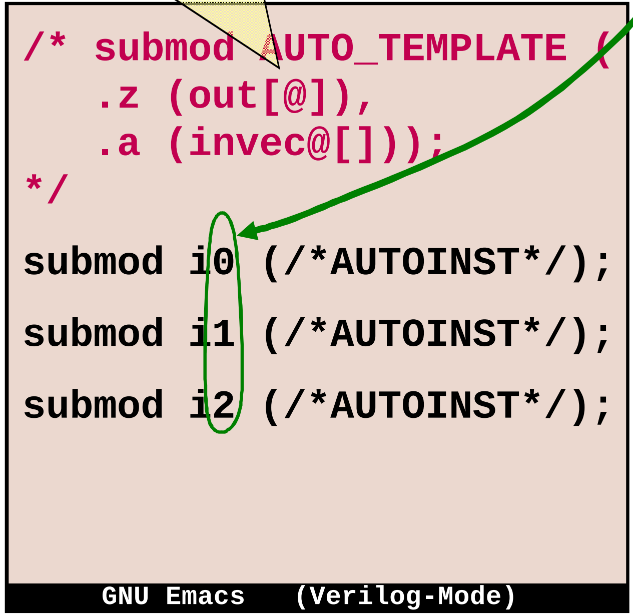
First time you're instantiating a module, let AUTOINST expand everything. Then cut the lines it inserted out, and edit them to become the template or exceptions.

```
/* 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

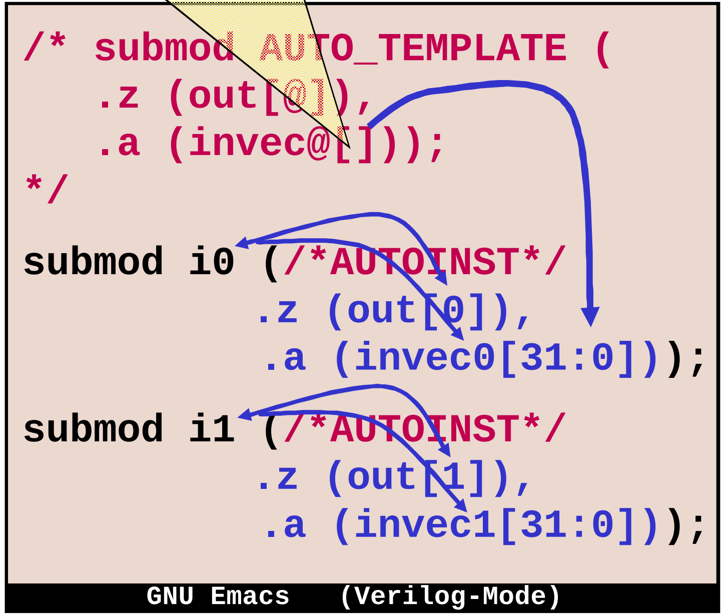
@ in the template takes the leading digits from the reference. (Or next slide.)



```
/* 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 [].



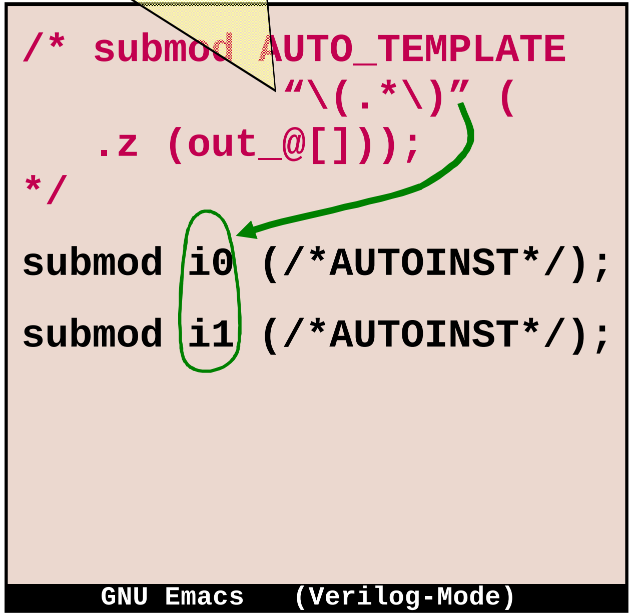
```
/* submod AUTO_TEMPLATE (
    .z (out[@]),
    .a (invec@[]));
*/
submod i0 (/*AUTOINST*/
           .z (out[0]),
           .a (invec0[31:0]));
submod i1 (/*AUTOINST*/
           .z (out[1]),
           .a (invec1[31:0]));
```

GNU Emacs   (Verilog-Mode)

Verilog Mode Emacs

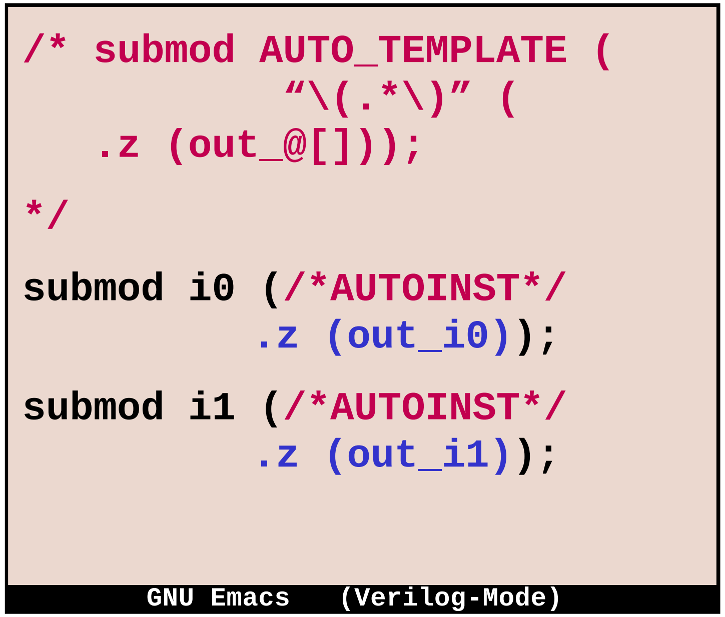
Overriding @

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)



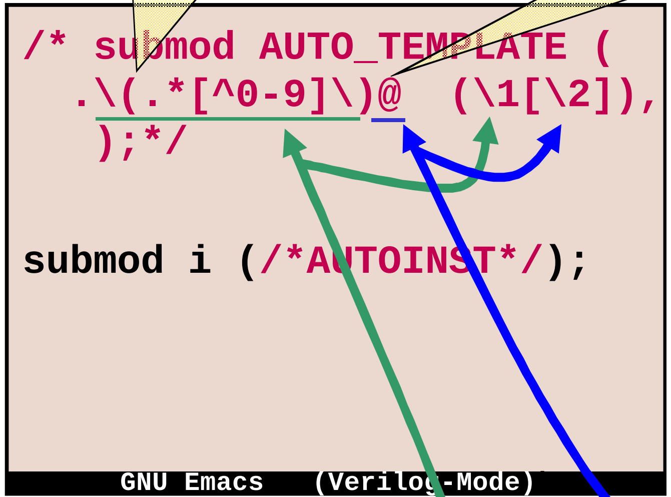
```
/* submod AUTO_TEMPLATE (
           "\(.*\)" (
   .z (out_@[]));

*/

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

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

GNU Emacs (Verilog-Mode)

Instantiations using RegExps

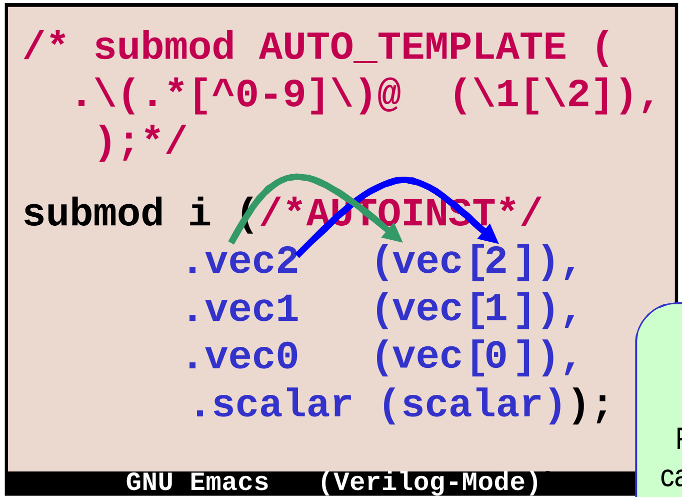
.\(\) indicates a Emacs regular expression.

@ 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.

Lisp Templates

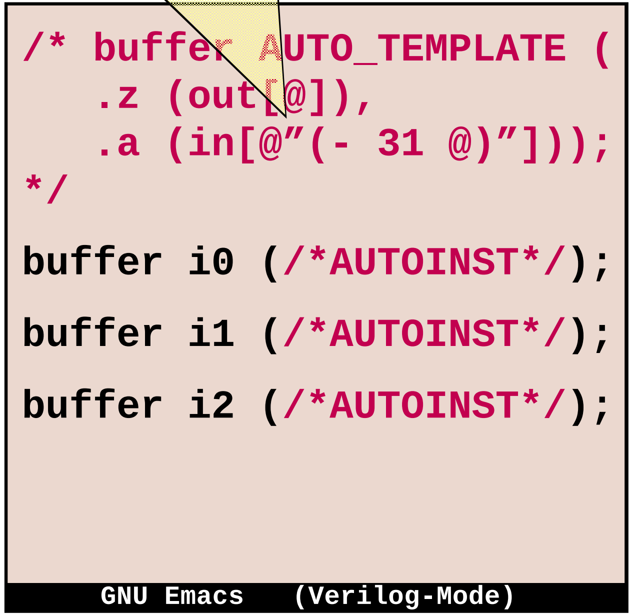
For even more complicated cases, see the documentation on Lisp templates

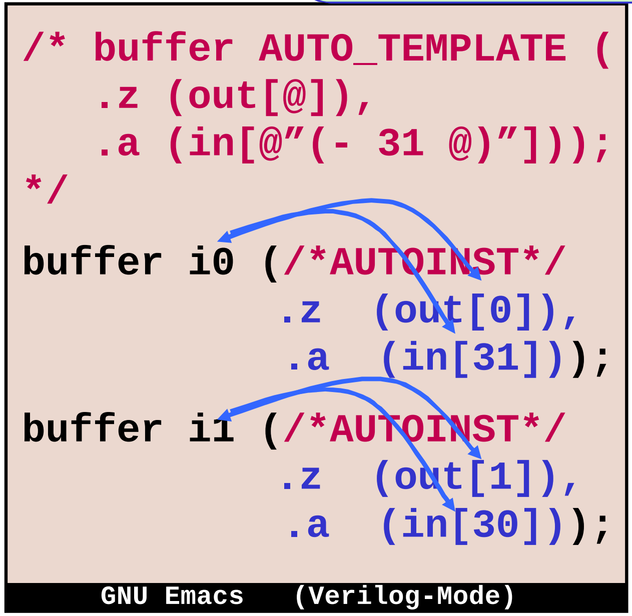
Instantiations using LISP

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

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*/);

buffer i1 (/*AUTOINST*/);

buffer i2 (/*AUTOINST*/);
```
GNU Emacs (Verilog-Mode)


```
/* 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 with Parameters

AUTOINSTPARAM is similar to AUTOINST, but “connects” parameters.

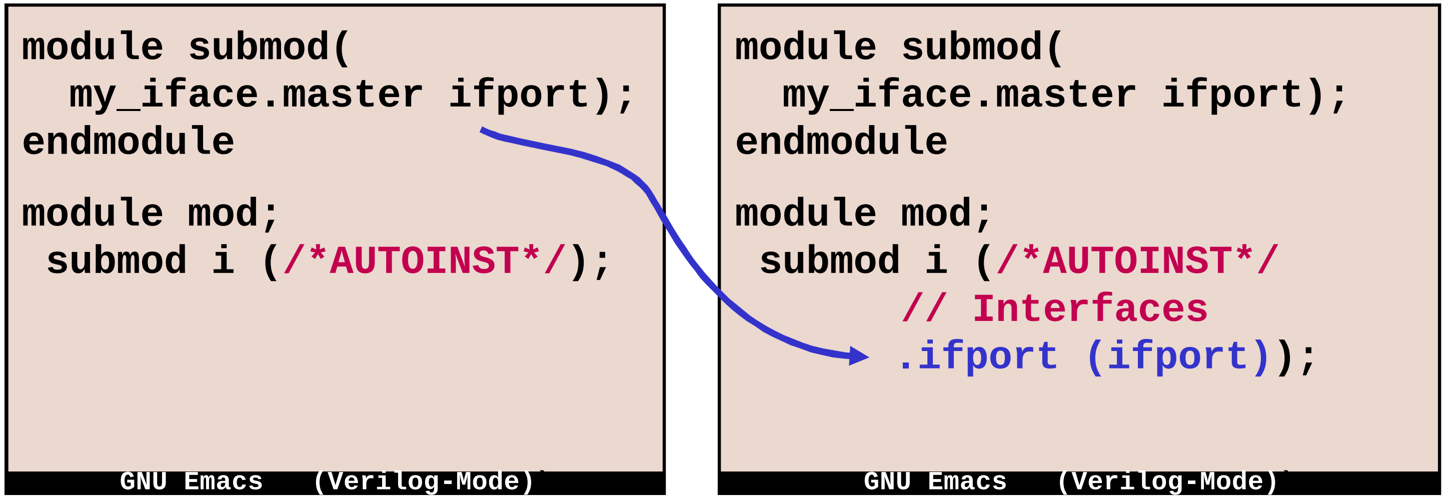
```
submod #(/*AUTOINSTPARAM*/)
    i (/*AUTOINST*/);
```

Regexp of desired parameters

```
submod #(/*AUTOINSTPARAM(“DEPTH”)*/)
    b (/*AUTOINST*/);

           GNU Emacs    (Verilog-Mode)
```

```
submod #(/*AUTOINSTPARAM*/
         .DEPTH(DEPTH),
         .WIDTH(WIDTH))
    i (/*AUTOINST*/
      …);

submod #(/*AUTOINSTPARAM(“DEPTH”)*/)
         .DEPTH(DEPTH));
    b (/*AUTOINST*/
      …);

           GNU Emacs    (Verilog-Mode)
```

Interfaces



```
module submod(
  my_iface.master ifport);
endmodule

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

GNU Emacs (Verilog-Mode)

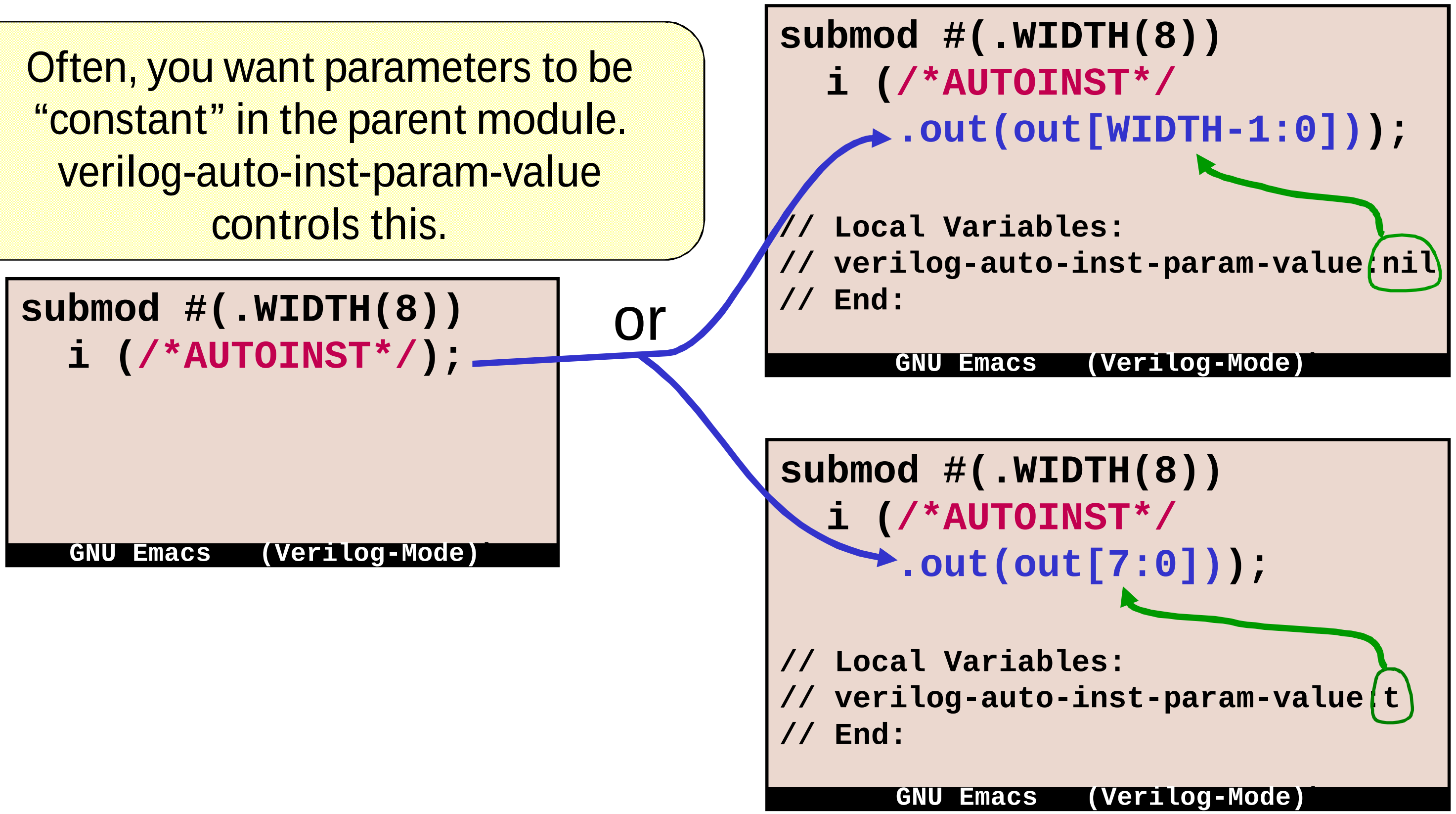
```
module submod(
  my_iface.master ifport);
endmodule

module mod;
 submod i (/*AUTOINST*/
      // Interfaces
      .ifport (ifport));
```

GNU Emacs (Verilog-Mode)

Verilog Mode Emacs

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)
```

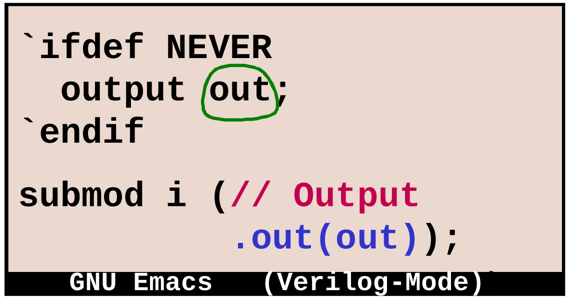
```
submod #(.WIDTH(8))
  i (/*AUTOINST*/
     .out(out[WIDTH-1:0]));

// 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

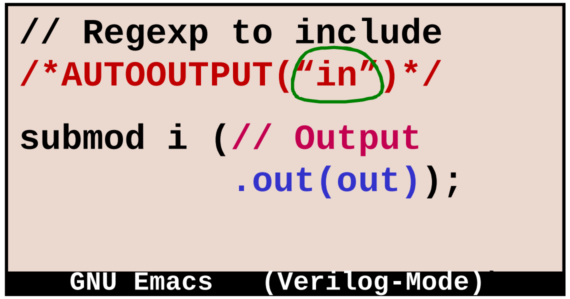
Techniques to exclude signals from AUTOOUTPUT (or AUTOINPUT etc).

1. Declare “fake” output



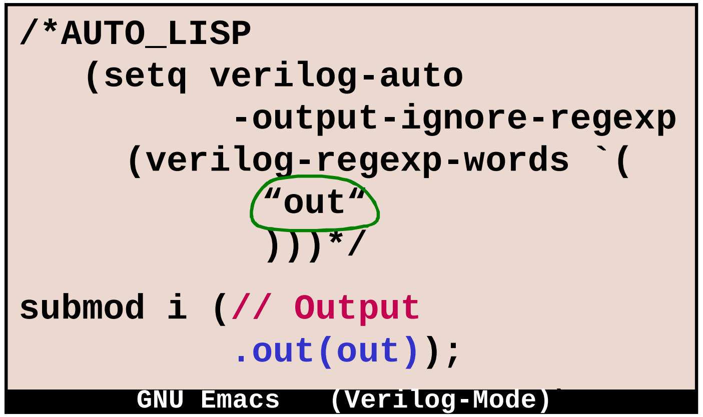
```
`ifdef NEVER
  output out;
`endif

submod i (// Output
          .out(out));
 GNU Emacs    (Verilog-Mode)
```

2. Use inclusive regexp



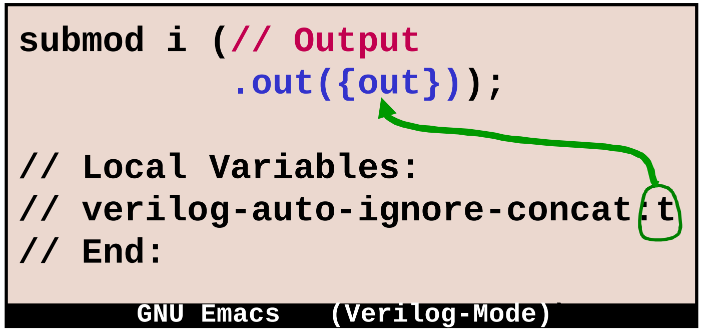
```
// Regexp to include
/*AUTOOUTPUT("in")*/

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)
```

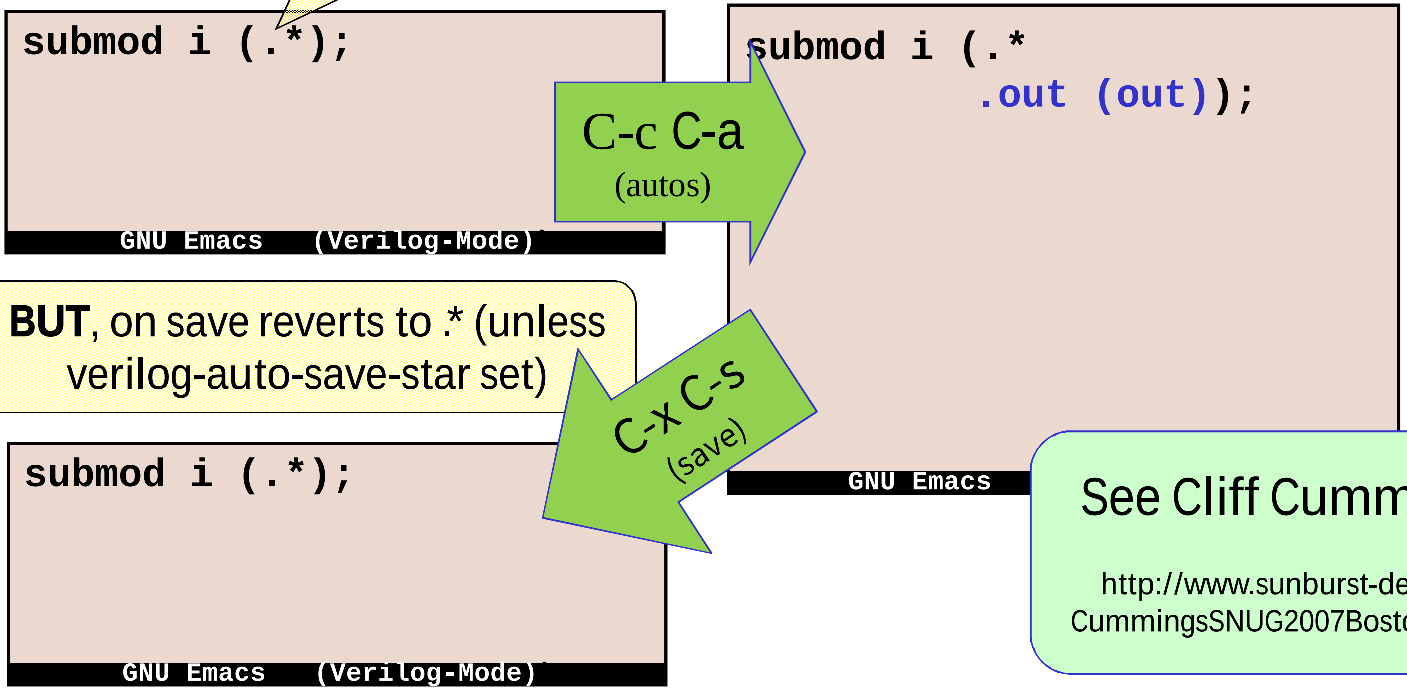
4. Use concats to indicate exclusion



```
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 (.*);
```

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

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

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

See Cliff Cumming's Paper

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

Where to Find Modules

1. Acceptable

```
// Local Variables:
// verilog-library-flags:("–y dir1 –y dir2")
// End:
                    GNU Emacs    (Verilog-Mode)
```

2. Best

```
// Local Variables:
// verilog-library-flags:("–f ../../input.vc")
// End:
                    GNU Emacs    (Verilog-Mode)
```

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

Use same input.vc as you feed to lint/synth/simulator

Jumping: C-c C-d

C-c C-d jumps to the definition of the entered module's name

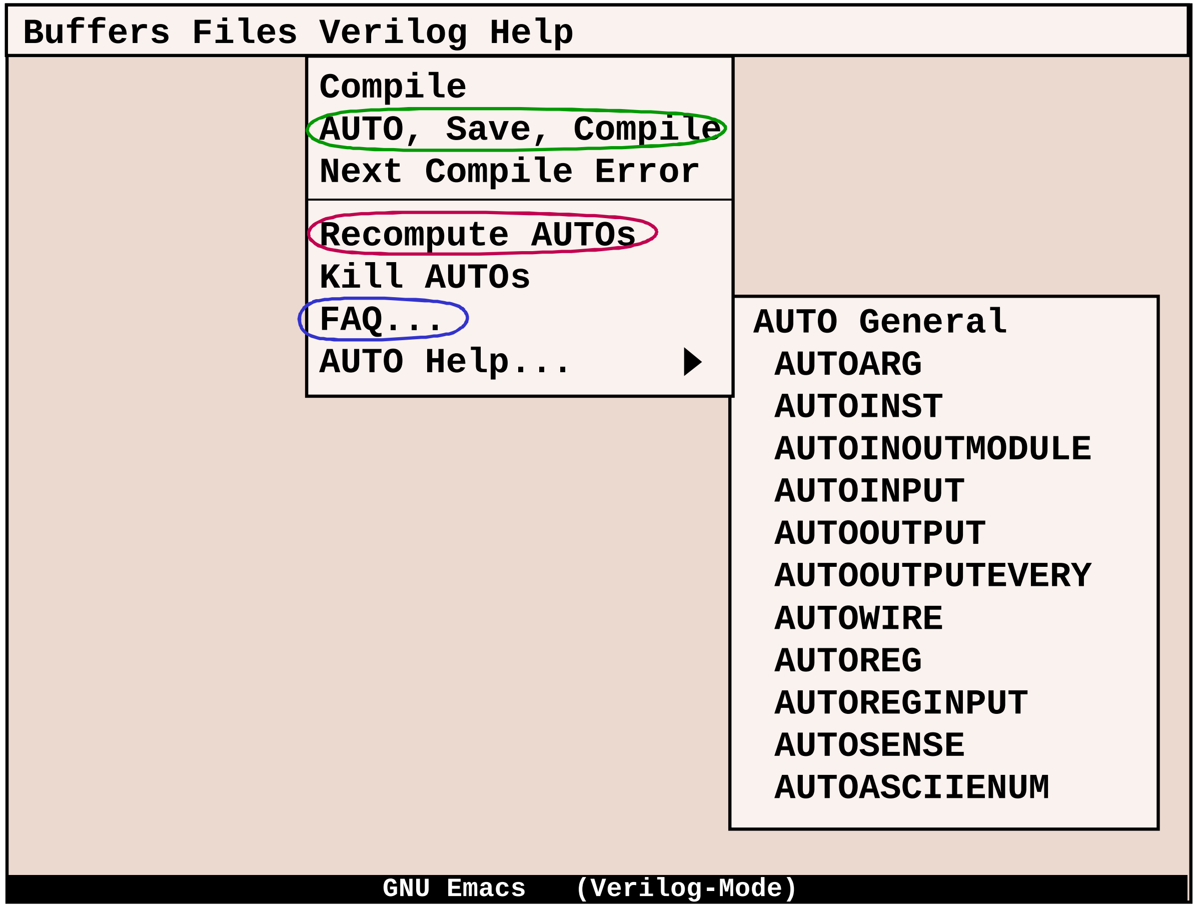
Agenda

- The Tedium of SystemVerilog
  - 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

Tips for Large Deployments

- In regressions, make sure code stays fresh & AUTOable
  - `verilog-batch-auto`
    - Make script to update your whole tree
  - `verilog-batch-diff-autos`
    - Add regression lint check that are no stale AUTOs
  - `verilog-auto-template-warn-unused`
    - Add regression lint check that are no stale AUTO_TEMPLATEs
  - `verilog-batch-indent`
    - Reindent all code – for Google style forced indentation methodology

Verilog Menu Help



Also see
https://www.veripool.org/verilog-mode-faq.html

Homework Assignment

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

Open Source

- Distributed with GNU Emacs
  - But usually years out of date, so install sources
- https://www.veripool.org
  - Git repository
  - Bug Reporting there via github (Please!)
  - These slides at https://www.veripool.org/papers/
- Additional Tools
  - Verilog-Perl – Toolkit with Preprocessing, Renaming, etc
  - Verilator – Compile SystemVerilog into C++/SystemC, also Python coming soon!

Verilog Mode

Emacs