Verilog-Mode

Reducing the Veri-Tedium

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

Last updated January, 2006

Agenda

- The Tedium of Verilog
  - What do I mean by Tedium?
  - Why bother to reduce it?
  - How do we reduce it?
- Verilog-mode Features
  - Sensitivity lists
  - Argument lists
  - Instantiations
  - Wires
  - Regs
  - State machines
- Getting it

Verilog Mode Emacs

Module Tedium?

```
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;

sub1 sub1 (.i1 (i1),
           .i2 (i2),
           .o2 (o2),
           .inter1 (inter1));

sub2 sub2 (.i1 (i1),
           .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.

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.

What would we like in a fix?

- Don’t want a new language
  - All tools would need a upgrade!
  - (Verilog 2000 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 input & output 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

Idea… Use comments!

Make /*AS*/ a special comment the program can look for.

```
always @(/*AS*/)
  begin
    if (sel) z = a;
    else     z = b;
  end
```

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

```
always @(/*AS*/
         a or b or sel)
  begin
    if (sel) z = a;
    else     z = b;
  end
```

```
always @(/*AS*/
         a or b or sel)
  begin
    if (sel) z = a;
    else     z = s2?c:d;
  end
```

If you then edit it, just rerun.

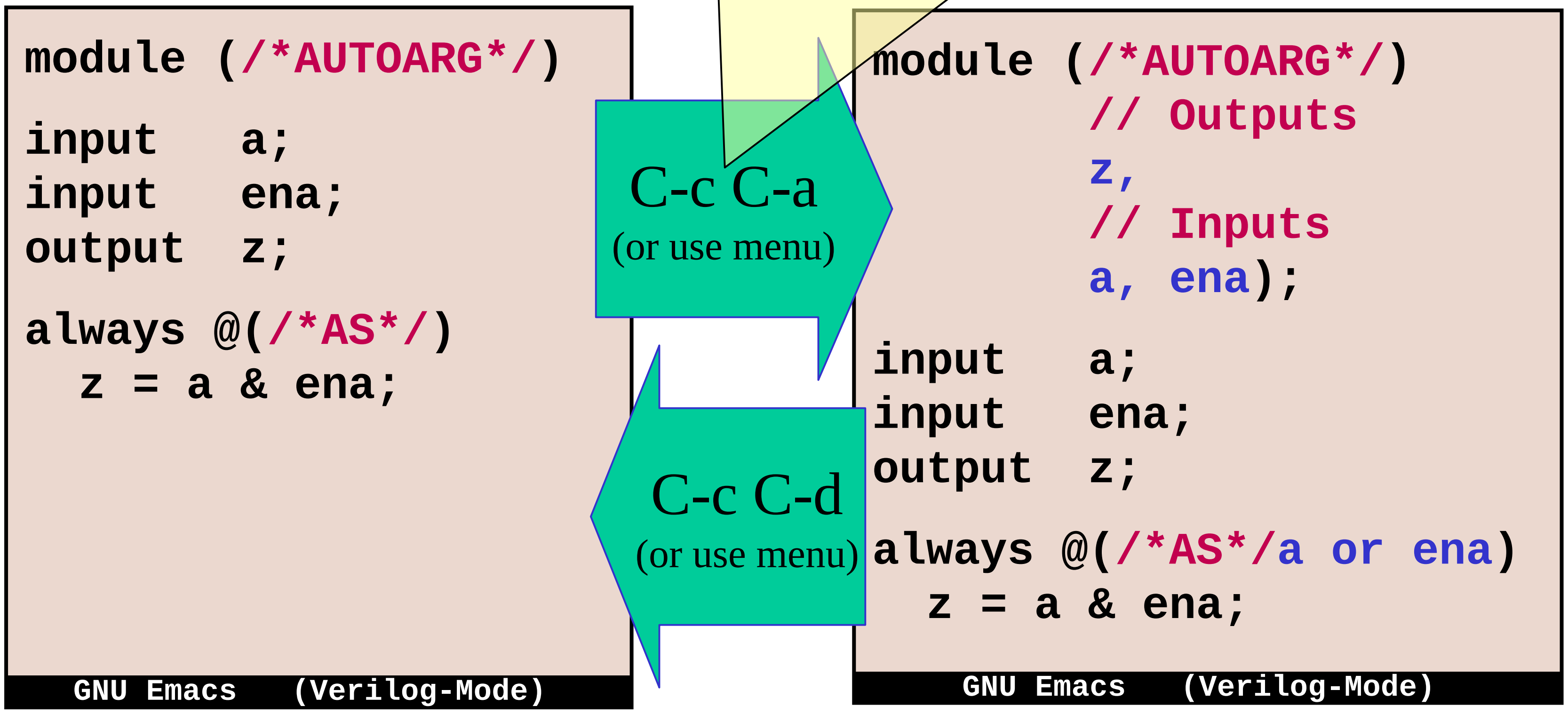
```
always @(/*AS*/
         a or c or d
         or s2 or sel)
  begin
    if (sel) z = a;
    else     z = s2?c:d;
  end
```

Extend Verilog-Mode for Emacs

- This expansion is best if in the editor
  - You can “see” the expansion and edit as needed
- There is a Verilog package for Emacs
  - Written by Michael McNamara `<mac@versity.com>`
  - Auto highlighting of keywords
  - Standardized indentation
- Expanded it to read & expand /*AUTOs*/
  - Magic key sequence for expand/deexpand

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.



```
module (/*AUTOARG*/)

input   a;
input   ena;
output  z;

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

`GNU Emacs    (Verilog-Mode)`

```
module (/*AUTOARG*/)
        // Outputs
        z,
        // Inputs
        a, ena);

input   a;
input   ena;
output  z;

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

`GNU Emacs    (Verilog-Mode)`

But the vi users revolt!

- ~~Too bad! Learn it.~~

Error!!!! Politically Incorrect

Error!!!! Starting a flame war

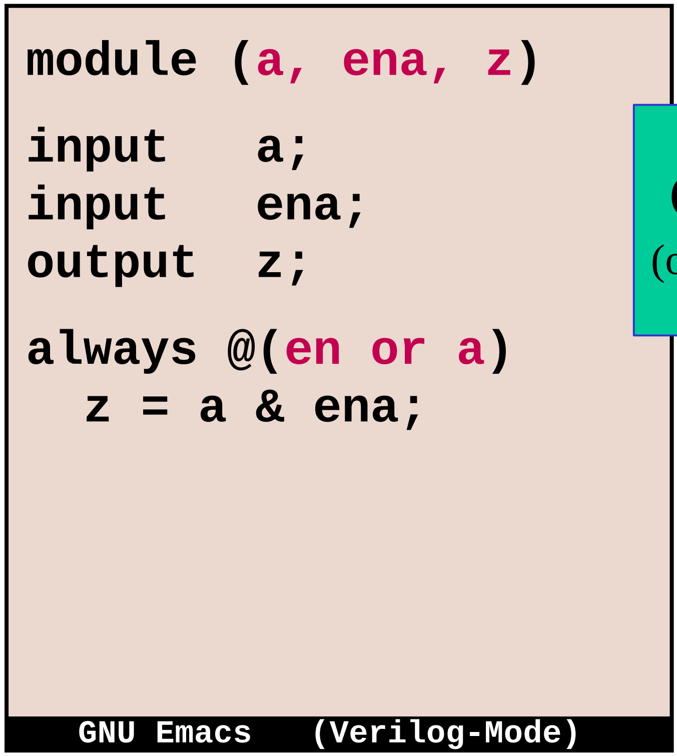
But the vi users revolt!

- Call Emacs as part of your flow/other editor
  - `emacs -batch {filename.v}`
    ```
    emacs -batch {filename.v}
         -f verilog-auto -f save-buffer
    ```
- Alternatively, the code is always valid!
  - Anyone using “edlin” or above 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 Features

C-c C-z: Inject AUTOs

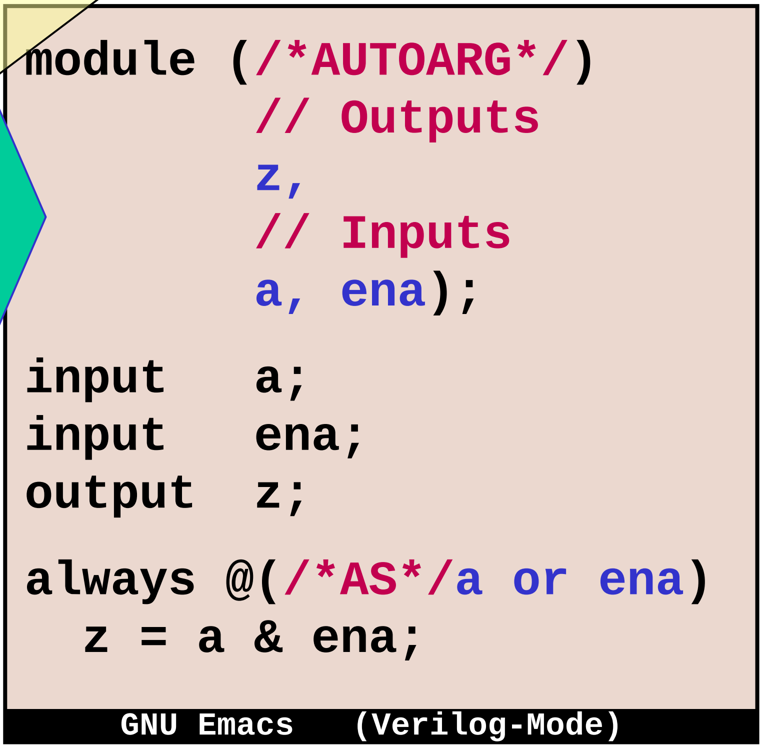
With this key sequence,
Verilog-Mode adds /*AUTOs*/ to old designs!



```
module (a, ena, z)

input   a;
input   ena;
output  z;

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

GNU Emacs   (Verilog-Mode)

C-c C-z
(or use menu)



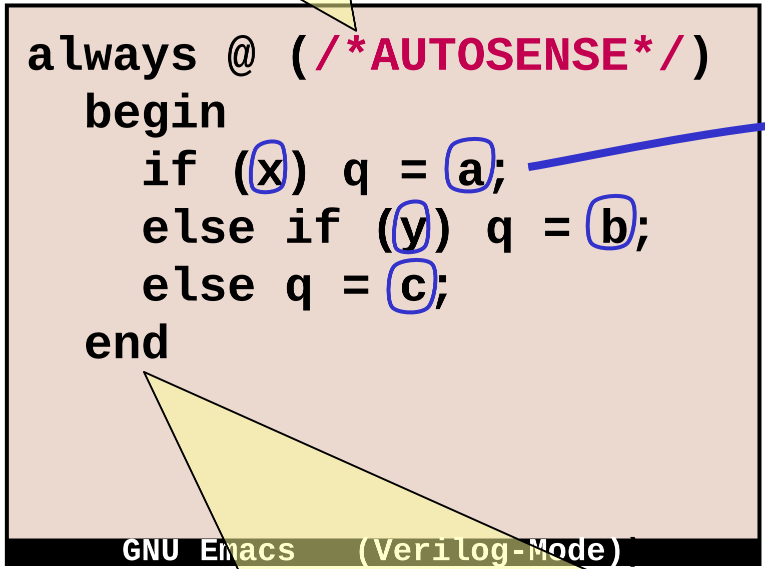
```
module (/*AUTOARG*/)
        // Outputs
        z,
        // Inputs
        a, ena);

input   a;
input   ena;
output  z;

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

GNU Emacs   (Verilog-Mode)

Sensitivity Lists

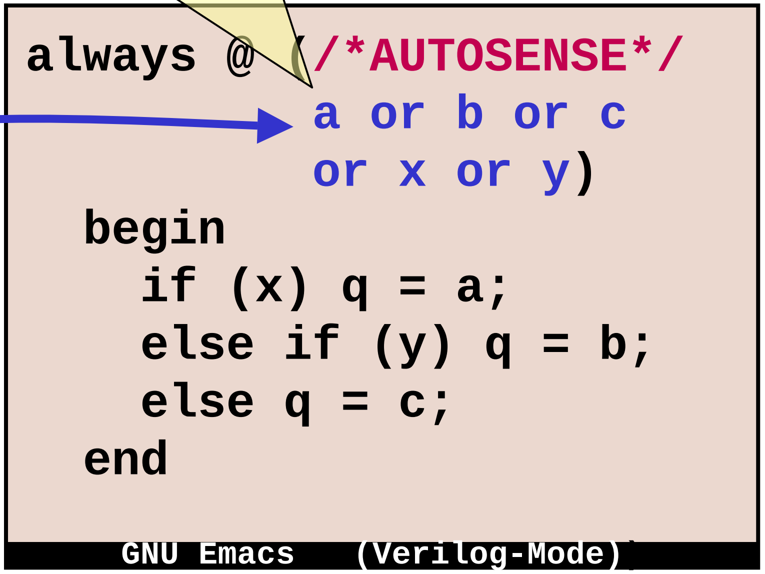
Alternatively, type /*AS*/ short for /*AUTOSENSE*/

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



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

GNU Emacs   (Verilog-Mode)



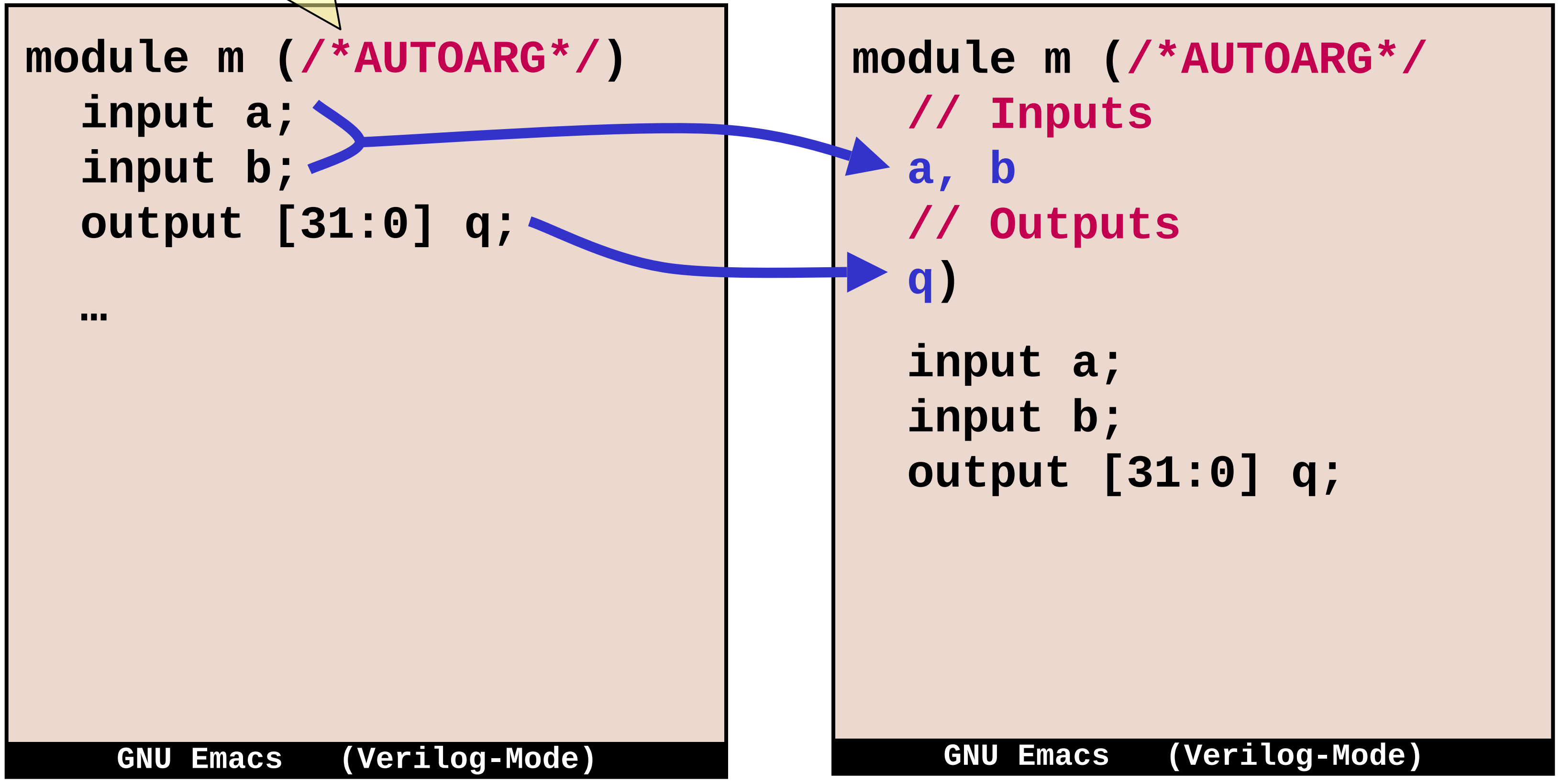
```
always @ (/*AUTOSENSE*/
          a or b or c
          or x or y)
   begin
      if (x) q = a;
      else if (y) q = b;
      else q = c;
   end
```

GNU Emacs   (Verilog-Mode)

Begin/end (or case/endcase) pairs after a always are a good idea. Verilog-mode can otherwise be confused in more complex cases. It also prevents a compile error if you add logic later (forgetting there’s no begin).

Argument Lists

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



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

   …
```

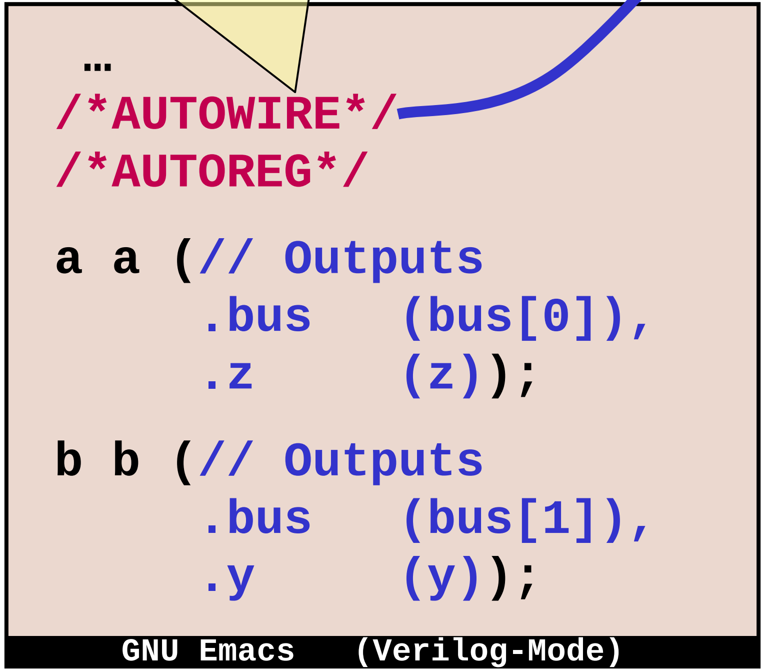
GNU Emacs   (Verilog-Mode)

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

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

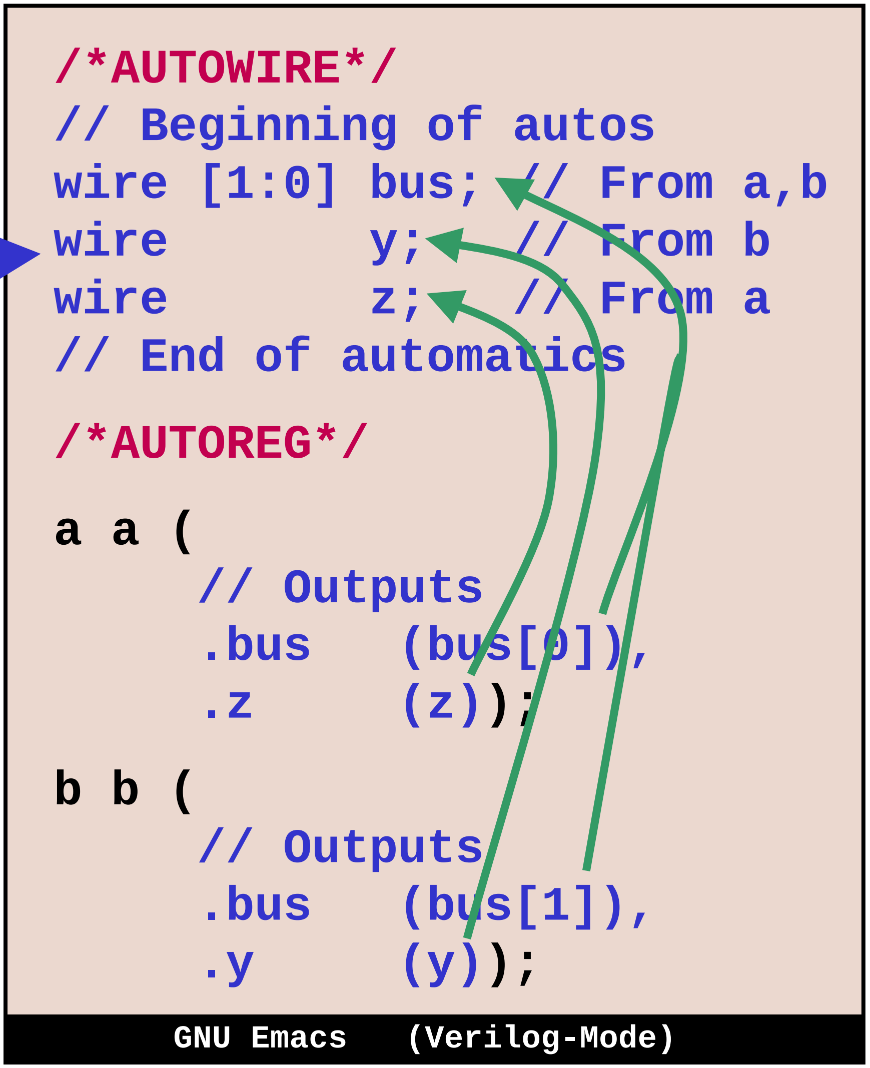
GNU Emacs   (Verilog-Mode)

Automatic Wires

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

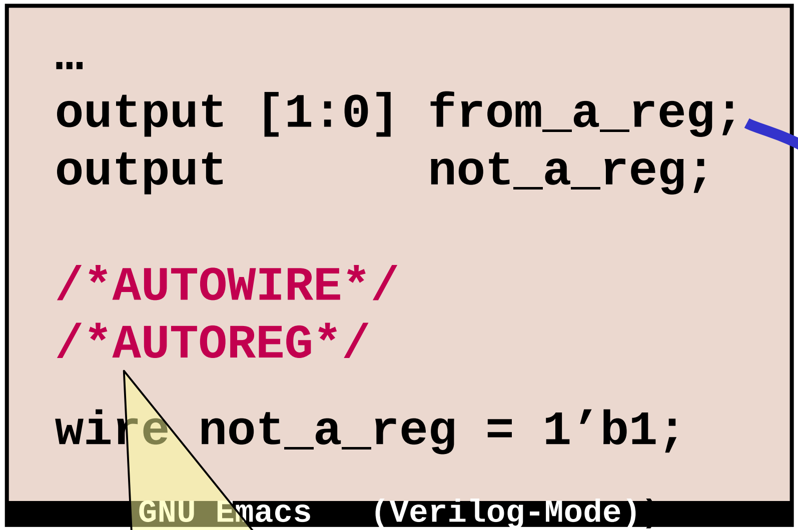


```
 …
/*AUTOWIRE*/
/*AUTOREG*/

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

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



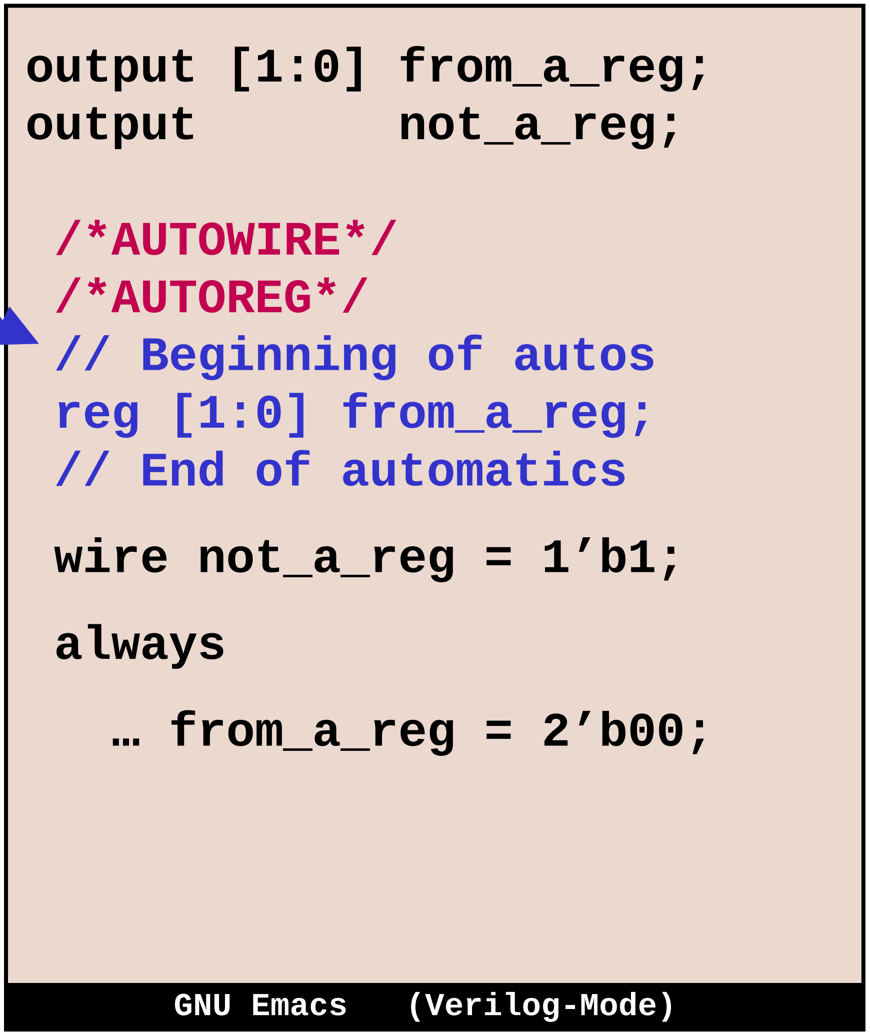
```
/*AUTOWIRE*/
// Beginning of autos
wire [1:0] bus; // From a,b
wire       y;   // From b
wire       z;   // From a
// End of automatics

/*AUTOREG*/

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

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

Automatic Registers



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

/*AUTOWIRE*/
/*AUTOREG*/

wire not_a_reg = 1'b1;
```

GNU Emacs   (Verilog-Mode)



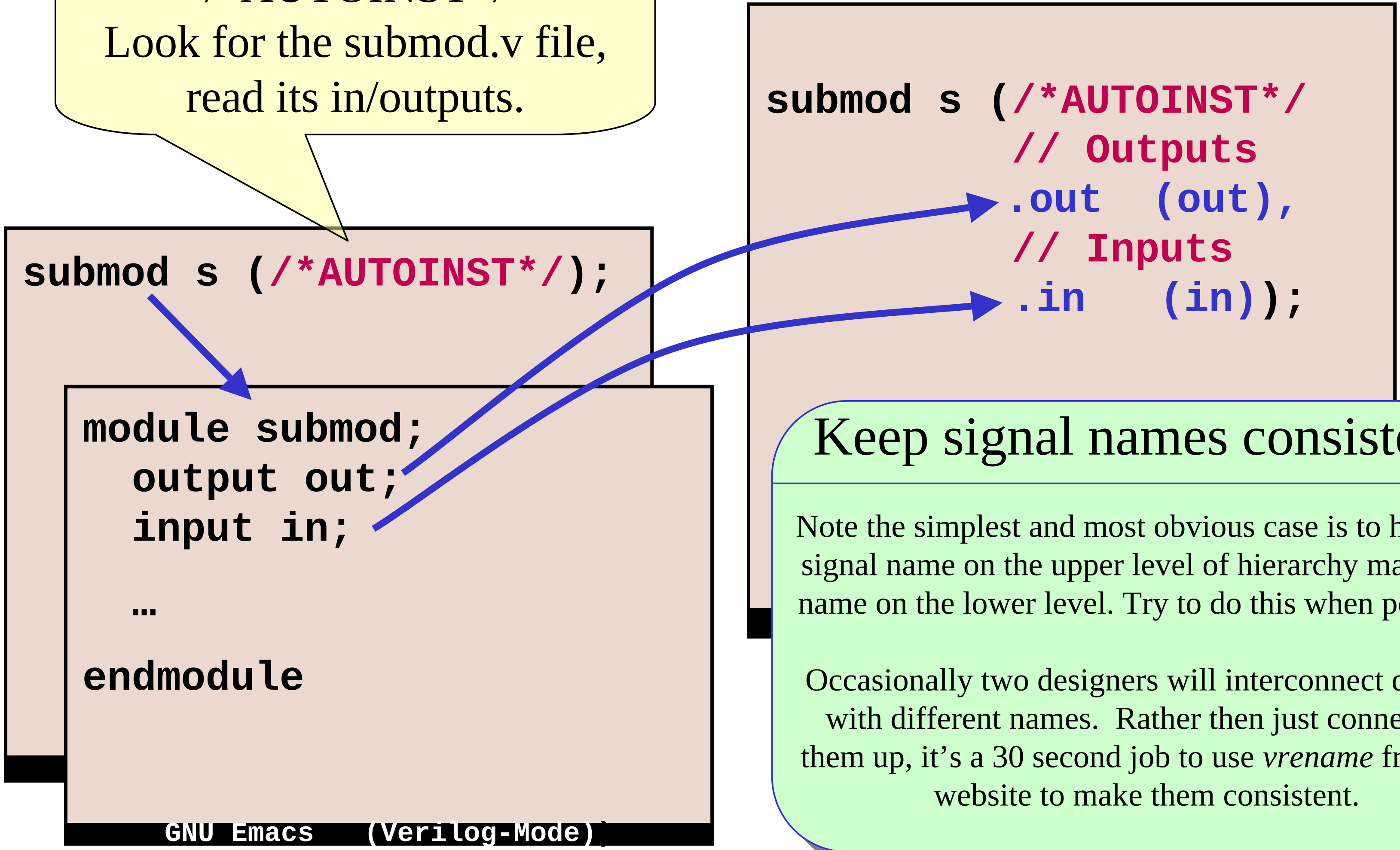
```
output [1:0] from_a_reg;
output       not_a_reg;

 /*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)

/*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.)

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 website to make them consistent.

Instantiation Example

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

  input  trdy;

  …
```

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

  input  irdy;

  …
```

```
module pci (/*AUTOARG*/);

 input irdy;
 input trdy;
 /*AUTOWIRE*/




 pci_mas mas (/*AUTOINST*/);




 pci_tgt tgt (/*AUTOINST*/);
```

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));
```

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;
  …
```

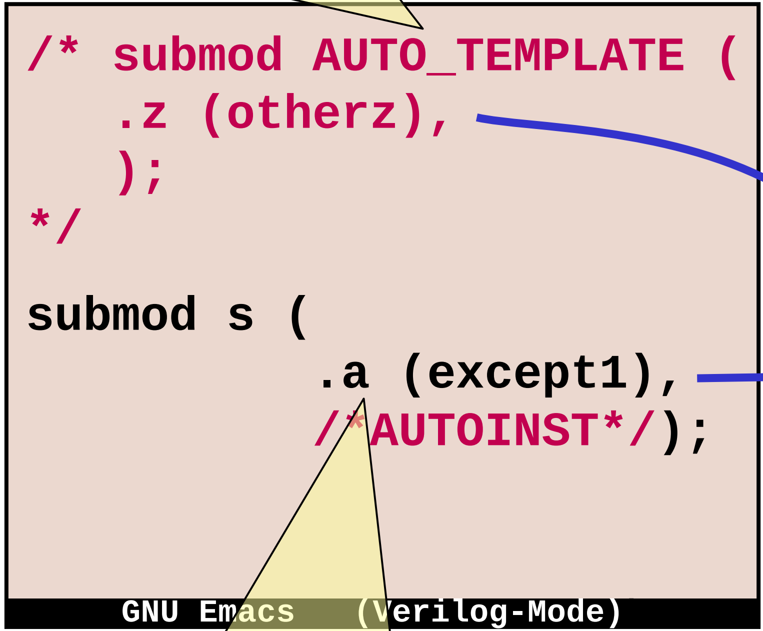
```
module pci (/*AUTOARG*/
            irdy, trdy);
 input irdy;
 input trdy;
 /*AUTOWIRE*/
 // Beginning of autos
 wire mas_busy;   // From mas.v
 // End of automatics

 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: A AUTO_TEMPLATE lists exceptions for “submod.” The ports need not exist.
(This is better if submod occurs many times.)

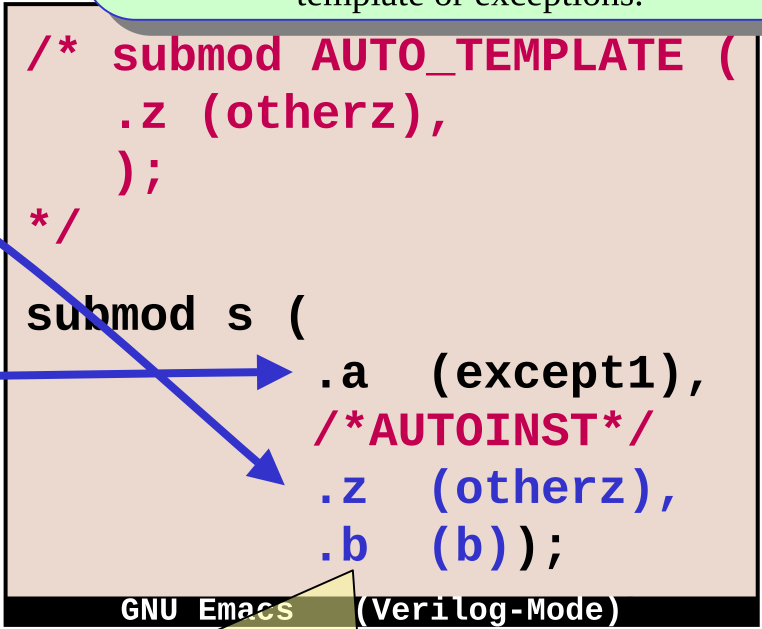
Initial Technique

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 (
          .a (except1),
          /*AUTOINST*/);
```

GNU Emacs (Verilog-Mode)



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

submod s (
          .a  (except1),
          /*AUTOINST*/
          .z  (otherz),
          .b  (b));
```

GNU Emacs (Verilog-Mode)

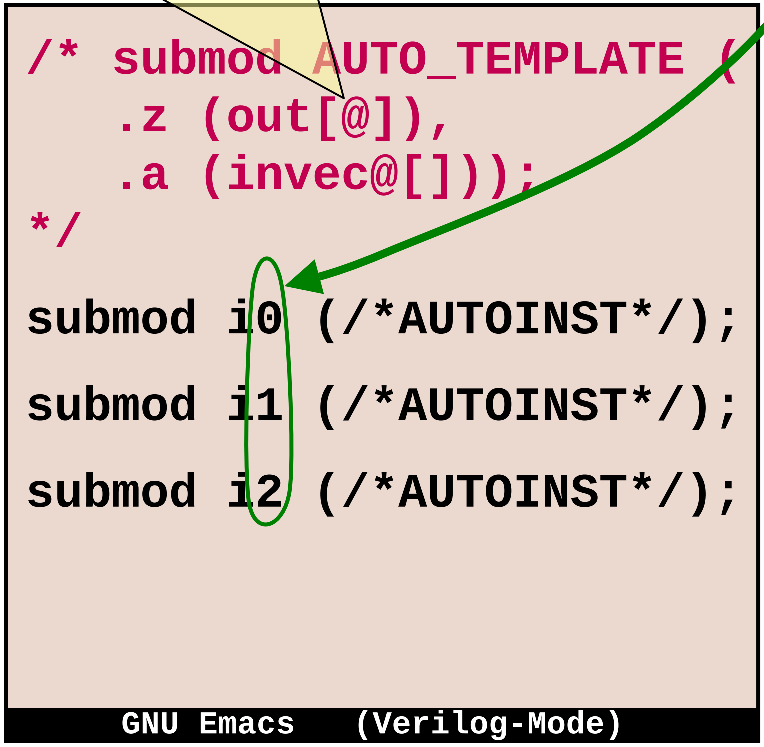
Method 2: List the signal before the AUTOINST.

Signals not mentioned otherwise are direct connects.

Multiple Instantiations

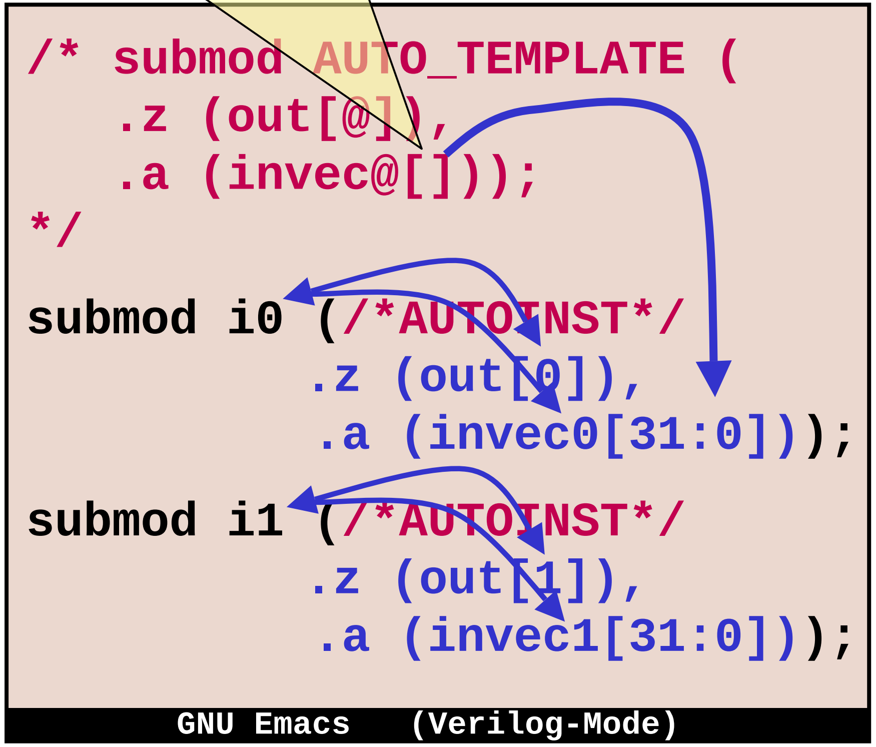
@ in the template takes the leading digits from the reference.

[] takes the bit range for the bus from the referenced module.



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

submod i0 (/*AUTOINST*/);

submod i1 (/*AUTOINST*/);

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

GNU Emacs (Verilog-Mode)

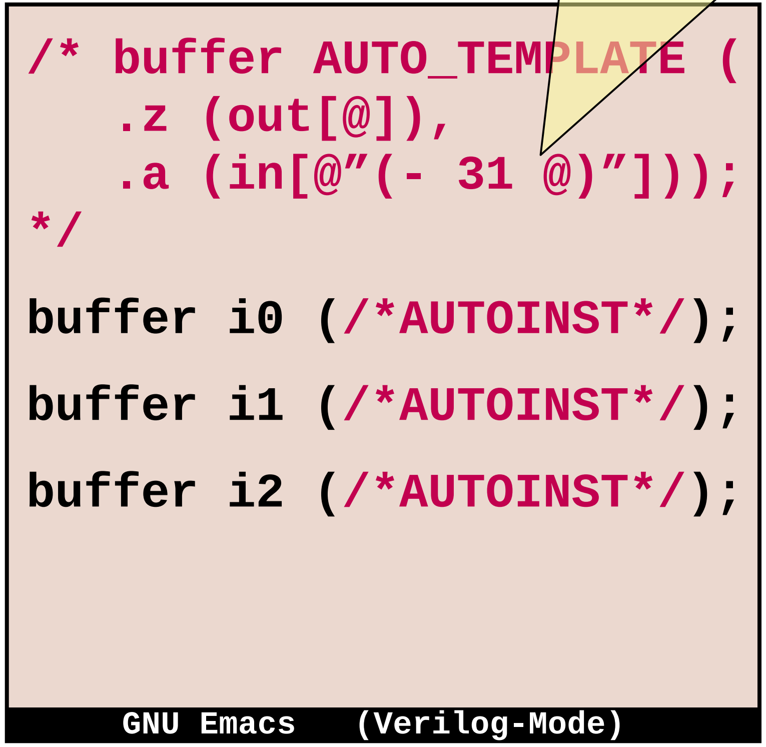


```
/* submod AUTO_TEMPLATE (
    .z (out[@]),
    .a (invec@[]));
*/

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

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

GNU Emacs (Verilog-Mode)

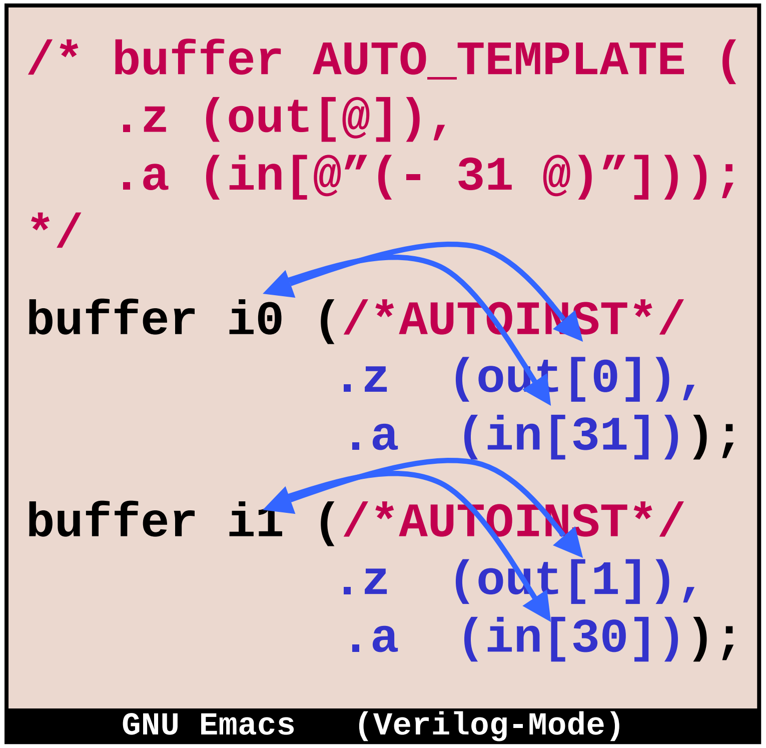
Instantiations using LISP

@"{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)



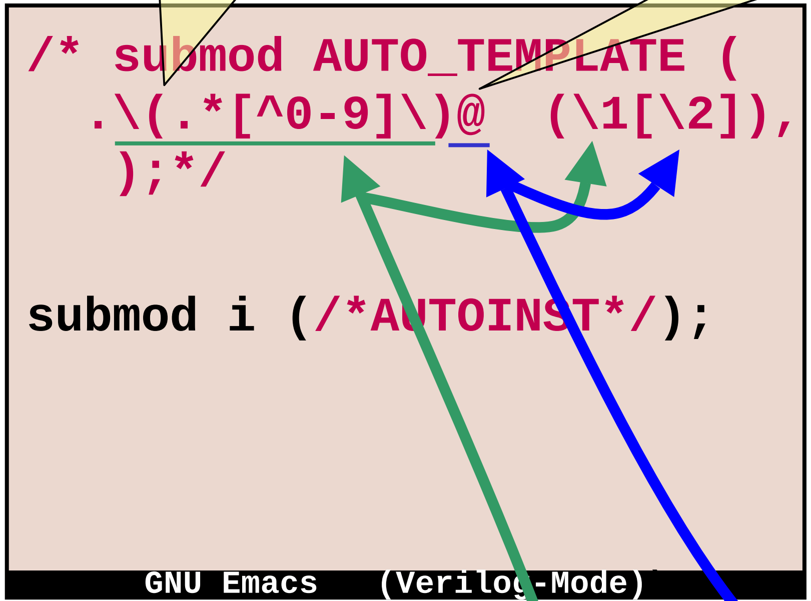
```
/* buffer AUTO_TEMPLATE (
   .z (out[@]),
   .a (in[@"(- 31 @)"]));
*/

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

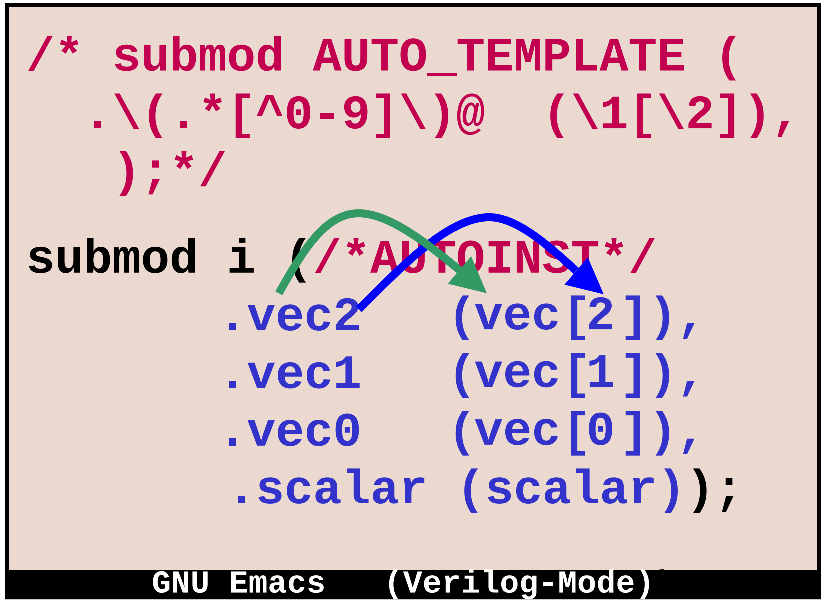
.\(\) indicates a Emacs regular expression.

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



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

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

GNU Emacs   (Verilog-Mode)



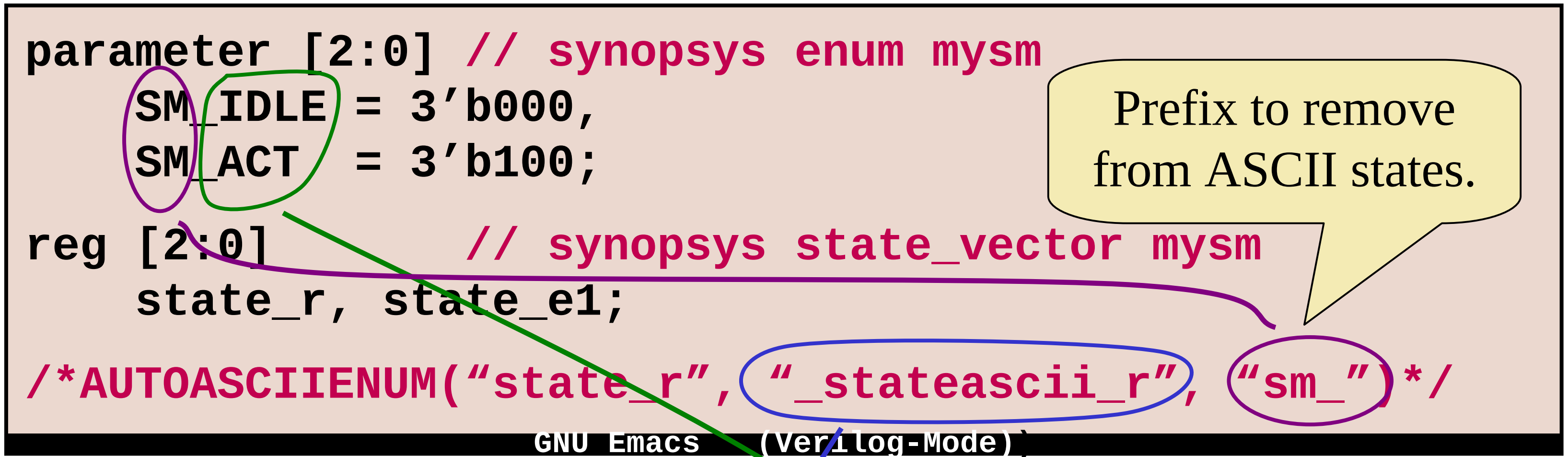
```
/* submod AUTO_TEMPLATE (
   .\(.*[^0-9]\)@  (\1[\2]),
   );*/

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.

State Machines

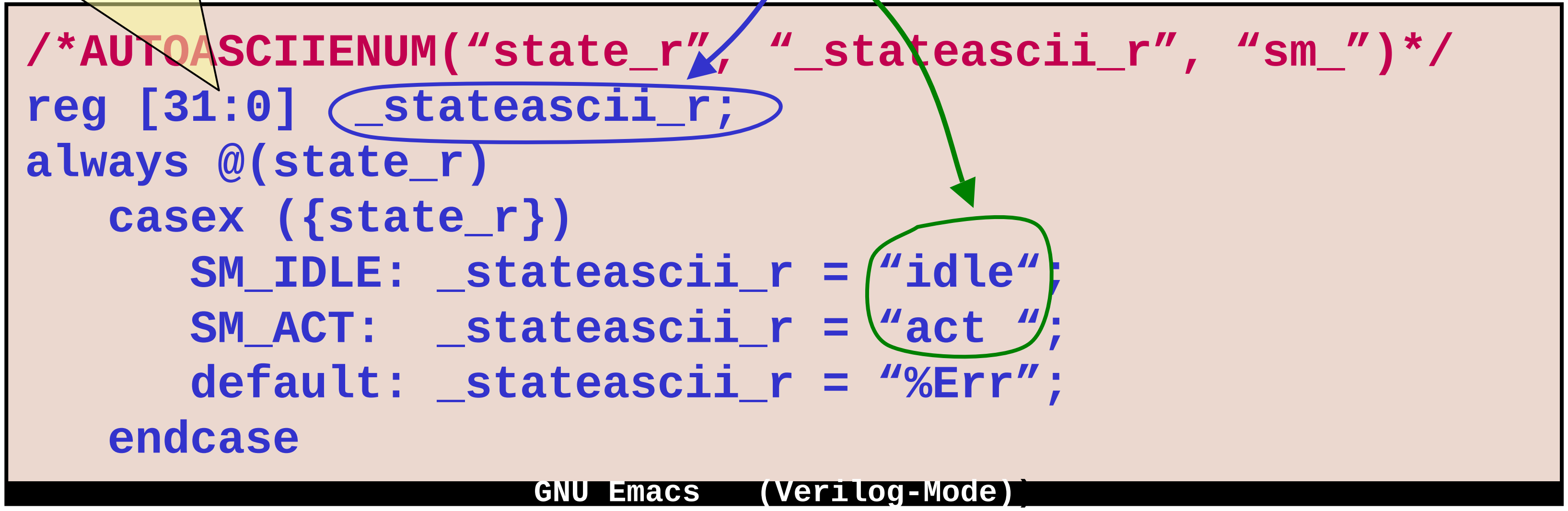


```
parameter [2:0] // synopsys enum mysm
    SM_IDLE = 3'b000,
    SM_ACT  = 3'b100;

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)
   casex ({state_r})
      SM_IDLE: _stateascii_r = "idle";
      SM_ACT:  _stateascii_r = "act ";
      default: _stateascii_r = "%Err";
   endcase
```

GNU Emacs (Verilog-Mode)

`ifdefs

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

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

   input a;
   input b;

`ifdef c_input
   input c;
`endif
```

GNU Emacs (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*/
   // 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.)

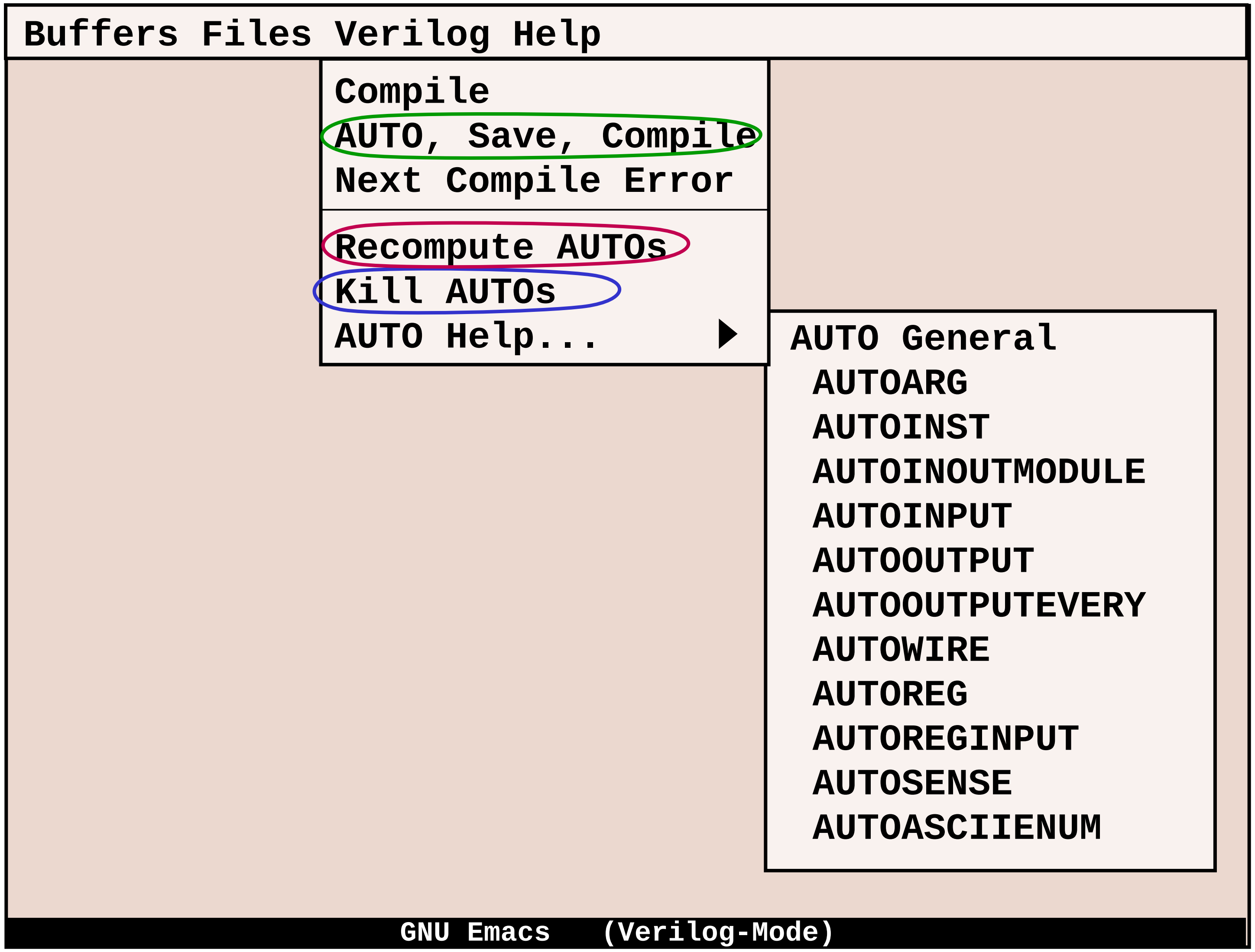
One ifdef would work, but consider multiple nested ifdefs each on overlapping signals. The algorithm gets horribly complex for the other commands (AUTOWIRE).

Making upper level modules

- Building null or shell modules
  - You want a module with same input/output list as another module.
  - /*AUTOINOUTMODULE(“*from.v*”)*/

- Output all signals
  - You have a shell which outputs everything.
  - /*AUTOOUTPUT*/
  - Dc_shell preserves output net names, so this is great for determining how fast each internal signal is generated.

Verilog Menu Help



```
Buffers Files Verilog Help

              Compile
              AUTO, Save, Compile
              Next Compile Error

              Recompute AUTOs
              Kill AUTOs
              AUTO Help...          ▶   AUTO General
                                          AUTOARG
                                          AUTOINST
                                          AUTOINOUTMODULE
                                          AUTOINPUT
                                          AUTOOUTPUT
                                          AUTOOUTPUTEVERY
                                          AUTOWIRE
                                          AUTOREG
                                          AUTOREGINPUT
                                          AUTOSENSE
                                          AUTOASCIIENUM

GNU Emacs   (Verilog-Mode)
```

Homework Assignment

- Homework
  - Due Next week:
    - Install Verilog-Mode
    - Use AUTOARG in one module
  - Grow from there!

Getting Verilog-Mode

- Homework
  - Due Next week: Use AUTOARG in one module
- GNU Licensed!
- Download site
  - You probably already have it! (Comes with VCS.)
  - http://www.veripool.com
- Contacting me

  `<wsnyder@wsnyder.org>`

Also at veripool.com

- Public Domain at http://www.veripool.com
  - Dinotrace – VCD Waveform Viewer
  - Verilog-Mode – The subject of this talk!
  - Verilator – Synthesizable Verilog Simulator
  - Verilog-Pli – Allow $pli calls to perl interpreter
  - VPM – Assertion check preprocessor
    - Best technical paper, SNUG Boston 2000
  - Vrename – Rename signals across an entire design