Verilog-Mode AUTOS
Reducing the SystemVerilog Tedium

https://www.veripool.org/papers

Wilson Snyder
April 6, 2020
(Updated April 15, 2020)
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
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.

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

sub s1 (/!*AUTOINST*/);

{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!

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

**C-c C-Z** (or use menu)

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));

GNU Emacs   (Verilog-Mode)

With "$ emacs --batch file.sv -f verilog-batch-auto"

(or use menu)

(or use menu)
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.

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;
  ...

module m /*AUTOARG*/
  // Inputs
  a, b
  // Outputs
  q)
  input a;
  input b;
  output [31:0] q;
/*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));
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

/*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.)

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

wire not_a_reg = 1'b1;

always
  from_a_reg = 2'b00;
```
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 <=.

```verilog
logic [1:0] a;
always @(posedge clk)
  if (reset) begin
    fsm <= ST_RESET;
    /*AUTORESET*/
  end
  else begin
    a <= b;
    fsm <= ST_OTHER;
  end
end
```

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

```
module ModStub (
    /*AUTOINOUTMODULE*/
    /*AUTOWIRE*/
    /*AUTOREG*/
    /*AUTOTIEOFF*/
    wire _unused_ok = &{
        /*AUTOUNUSED*/
        1'b0};
endmodule
```

```
module ModStub (
    /*AUTOINOUTMODULE*/
    /*AUTOWIRE*/
    /*AUTOREG*/
    /*AUTOTIEOFF*/
    wire [2:0] mod_out = 3'b0;
    wire _unused_ok = &{
        /*AUTOUNUSED*/
        mod_in,
        1'b0};
endmodule
```
Script Insertions

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

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

Insert Lisp result.

Insert shell result.

/*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.

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

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

logic [2:0] // synopsys state_vector mysym
    state_r, state_e1;

/*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
```

Prefix to remove from ASCII states.

Sized for longest text.

Prefix to remove from ASCII states.
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
Simple Instantiations

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

```
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.
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

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

pci_mas mas
  (/*AUTOINST*/
   // Inputs
   .trdy     (trdy),
   // Outputs
   .mas_busy (mas_busy));
module pci_mas
  (/*AUTOARG*/
   trdy, mas_busy);
input  trdy;
output mas_busy;
endmodule

pci_tgt tgt
  (/*AUTOINST*/
   // Inputs
   .irdy     (irdy),
   .mas_busy (mas_busy));
module pci_tgt
  (/*AUTOARG*/
   irdy, mas_busy);
input  irdy;
input  mas_busy;
endmodule
```

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

pci_mas mas
  (/*AUTOINST*/
   // Inputs
   .trdy     (trdy),
   // Outputs
   .mas_busy (mas_busy));
module pci_mas
  (/*AUTOARG*/
   trdy, mas_busy);
input  trdy;
output mas_busy;
endmodule

pci_tgt tgt
  (/*AUTOINST*/
   // Inputs
   .irdy     (irdy),
   .mas_busy (mas_busy));
module pci_tgt
  (/*AUTOARG*/
   irdy, mas_busy);
input  irdy;
input  mas_busy;
endmodule
```
Exceptions to Instantiations

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

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

Signals not mentioned otherwise are direct connects.

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.
Multiple Instantiations

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

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

```verilog
/* submod AUTO TEMPLATE (  
  .z (out[@]),  
  .a (invec@[]));  
*/
submod i0 /*AUTOINST*/;
submod i1 /*AUTOINST*/;
submod i2 /*AUTOINST*/;
```

```verilog
/* submod AUTO TEMPLATE (  
  .z (out[8]),  
  .a (invec@[]));  
*/
submod i0 /*AUTOINST*/;
  .z (out[0]),
  .a (invec0[31:0]);
submod i1 /*AUTOINST*/;
  .z (out[1]),
  .a (invec1[31:0]);
```
A regexp after AUTO_TEMPLATE specifies what to use for @ instead of last digits in cell name. Below, @ will get name of module.

```verilog
/* submod AUTO_TEMPLATE
   \"\((.*\\)\)\" ( 
   .z (out_@[])); 
*/
submod i0 (/*AUTOINST*/);
submod i1 (/*AUTOINST*/);
```

```verilog
/* submod AUTO_TEMPLATE ( 
   \"\((.*\\)\)\" ( 
   .z (out_@[])); 
*/
submod i0 (/*AUTOINST*/
   .z (out_i0));
submod i1 (/*AUTOINST*/
   .z (out_i1));
```
Instantiations using RegExps

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}]

/* buffer AUTO_TEMPLATE (  
    .z (out[@]),  
    .a (in[@"(- 31 @)"]));  
*/
buffer i0 /*AUTOINST*/;
buffer i1 /*AUTOINST*/;
buffer i2 /*AUTOINST*/;

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.
Instantiations with Parameters

AUTOINSTPARAM is similar to AUTOINST, but "connects" parameters.

Regexp of desired parameters

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

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

```
submod #(/*AUTOINSTPARAM("DEPTH")*/)
    .DEPTH(DEPTH));
```

```
submod #(/*AUTOINSTPARAM("DEPTH")*/
    .DEPTH(DEPTH))
    ...
);
```
module submod(
    my_iface.master ifport);
endmodule

module mod;
    submod i (/*AUTOINST*/);
endmodule
Often, you want parameters to be “constant” in the parent module. verilog-auto-inst-param-value controls this.

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

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

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

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

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

1. Declare “fake” output
   ```verilog
   ifdef NEVER
     output out;
   endif
   submod i (// Output
     .out(out));
   ```

2. Use inclusive regexp
   ```verilog
   // Regexp to include
   /*AUTOOUTPUT("in")*/
   submod i (// Output
     .out(out));
   ```

3. Set the output ignore regexp
   ```verilog
   /*AUTO_LISP
    (setq verilog-auto
     -output-ignore-regexp
     (verilog-regexp-words `(\"out\")*))*/
   submod i (// Output
     .out(out));
   ```

4. Use concats to indicate exclusion
   ```verilog
   submod i (// Output
     .out({out}));
   ```
SystemVerilog .*

SystemVerilog .* expands just like AUTOINST.

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

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

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

C-c C-a (autos)

C-x C-s (save)

See Cliff Cumming’s Paper

Where to Find Modules

1. Acceptable

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

2. Best

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

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

Buffers Files Verilog Help

Compile
AUTO, Save, Compile
Next Compile Error
Recompute AUTOs
Kill AUTOs
FAQ...
AUTO Help...

AUTO General
AUTOARG
AUTOINST
AUTOINOUTMODULE
AUTOINPUT
AUTOOUTPUT
AUTOOUTPUTEVERY
AUTOWIRE
AUTOREG
AUTOREGINPUT
AUTOSENSE
AUTOASCIIENUM

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!