Verilog Preprocessor: Force for `Good and `Evil

http://www.veripool.org/papers

Wilson Snyder
Cavium Networks
wsnyder@wsnyder.org

`Agenda

- Standards & Compliance
- Best Practices
- Applications
- Metaprogramming
- Preprocess For Free
- Conclusions
- Q&A

`Standards

Synopsys Users Group
BOSTON 2010

Verilog 1995
`define MACRO
`ifdef `else `endif
`include

Verilog 2001
`define MACRO(*args…*)
`ifndef `elsif `undef
`line

SystemVerilog 2005
`` `" `\`"

SystemVerilog 2009
`define MACRO(*arg=default…*)
`undefineall
`__FILE__ `__LINE__

Compared to "C"?

It's close, but there's gotchas – see the paper!

`Beware_Vendor_Compliance

Synopsys Users Group
BOSTON 2010

- Standards support:
  - 2001 define arguments is almost universal
  - 2001 `line is very good
  - 2005 `` and similar is very good, for SV tools
  - 2009 default arguments are almost non existent
  - ~2012 some of the tricks here?

- See the paper
  - VCS does very well
  - Wrap `" in a STRINGIFY macro

`Agenda

- Standards & Compliance
- Best Practices (the `good)
- Applications
- Metaprogramming
- Preprocess for Free
- Compliance
- Conclusions
- Q&A

`Parens

Synopsys Users Group
BOSTON 2010

- Beware mis-paren-itis in `define functions:

```
`define A(x,y) x|y
wire b = `A(a,b) & c;  //== a | b&c !!
wire b = `A(a&b, c);   //== a & b|c !!
```

wrong precedence!

- Solution?
  - Rule: Parenthesize formal arguments and the output

```
`define A(x,y) ((x)|(y))
 wire b = `A(a,b) & c;  //== ((a)|(b)) & c
 wire b = `A(a&b, c);   //== ((a&b))  | (c)
```

`Name_Collisions

Synopsys Users Group
BOSTON 2010

- `defines have global scope
  - And, many tools don't warn about conflicts:

```
`define A 1'b1  // in file1.vh
`define A 1'b0  // in file2.vh
```

- Solution?
  - Rule: Prefix all includes to make them unique
    - vrename can help fix this
  - Rule: If in used one file, undef at end of usage
    - Some lint tools can enforce this

```
`define FILE1_A 1'b0
... `FILE1_A ...
`undef FILE1_A
```

`Side_Effects

- Defines are substituted literally

```
`define D(x)  ((x)?f(x):0)
b = `D(y++)   //== ((y++)?f(y++):0) !!
```

"A side effect"

y increments twice!

- Solution?
  - Rule: Use formals only once per call
  - Rule: Avoid passing side effects into a macro
  - Suggestion: Use functions instead

`Task_Do_While

- Beware multi-statement task macros:

```
`define WARN(msg) \
  $write("Warn: "); $write(msg);
…
if (x) `WARN("x");
//->  if (x) $write("Warn:");
//->  $write("x");;  !!
```

Not part of if!

Extra semicolon; eliminate by removing

- Rule: Wrap "task" macros in do…while

```
`define WARN(msg) \
  do begin \
    $write("Warn: "); $write(msg); \
  end while(0)
```

`Replace_Metacomments

- Avoid metacomments

```
// synopsys translate_off
`MSG_MACRO(…)
// synopsys translate_on
```

- Solution?
  - Rule: Use `ifdef, not translation metacomments
  - Suggestion: Move synthesis ifdef into macro definition

```
`ifndef SYNTHESIS
  `define MSG_MACRO(m) …
`else
  `define MSG_MACRO(m) do while(0)
`endif
`MSG_MACRO(…)
```

“nop” using do-while trick

`Alternatives

- Metacomments vs `ifdef
  - Previous page
- Localparams vs `define
  - Module local
  - Allow bit extraction, though fixed with {`FOO}[1:0]
  - Different values per module can be a problem
  - Down to a style difference – for global stuff I use `define
- Custom Scripts vs `define hacks
  - For internal CSRs, memory generators, pads, etc custom code generator scripts are probably best
  - For IP, `defines are far more portable

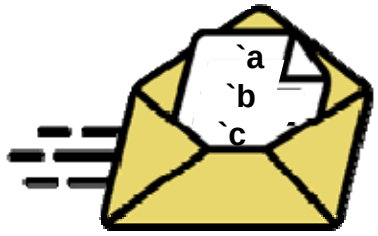
`Agenda

- Standards & Compliance
- Best Practices
- Applications
- Metaprogramming
- Preprocess for Free
- Conclusions
- Q&A

`Message_Macros

- Desire to wrap $display

```
`MSG(format)
```



- One Solution:

```
`define MSG(m) \
  do begin \
    $write("[%0t] %s:%d: %s\n", \
      $time, `__FILE__, `__LINE__, \
      $sformatf m) \
  while(0)
```

- If want to allow variable arguments, requires extra parens when used (see paper):

sv2012?

```
`MSG(("Hello %d world", 1))
```

`Assert_Macros

- Wrap assertions

```
`ASSERT_PROP(label,when)
```

  - Enable calling custom PLI function on failure
  - Allow adding `ifndef SYNTHESIS around the assertion

- One Solution:

```
`define _ASSERT_MSG(label) \
   `"Assertion 'label' failed`"
`ifndef SYNTHEIS
  `define ASSERT_PROP(label,when) \
     label : assert property (when) \
        else $err(`_ASSERT_MSG(label))
`else …
```

`" is Substitute-in-string

`Module_Selection

- Simulation wants to choose between “real” and “stub” module instantiations with identical pinout

- Solution?
  - Don’t use “config” – Not generally supported
  - Don’t use “generate if” – Error prone if pinout the same
  - Use an `ifdef

```
`ifdef MY_STUB_RAMS
  `define Ram RamStub
`else
  `define Ram Ram
`endif
`Ram ram (/*AUTOINST*/);
```

`Line_Generators

Synopsys Users Group
BOSTON 2010

- Programs that generate code should report errors relative to original input
  - Build process does a “cp input.v temp.v”
  - Compiler would report

```
Error: temp.v:123: Something bad
```

- Solution?
  - Use `line

```
echo ‘`line 1 “input.v” 0’ > temp.v
cat input.v >> temp.v
```

  - Now we get:

```
Error: input.v:123: Something bad
```

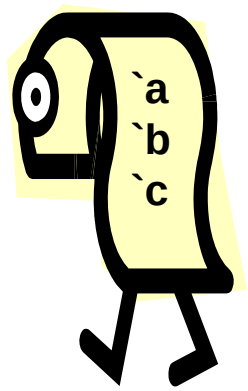
  - See paper for more practical examples

`Lists(1)

Synopsys Users Group
BOSTON 2010

- Repeated lists are hard to maintain:

```
typedef enum {
  OP_INST1,
  OP_INST2, …
} Op;
function bit opAlu(Op o) {
  case (o)
    OP_INST1: return 1;
    OP_INST2: return 0;
…
```



- A code generator script may be the best solution
- Or, use the preprocessor…

`Lists(2)

Synopsys Users Group
BOSTON 2010

- In a unique file, call a macro with each item

```
// ops.vh
//      name,  iop, alu
`INSTR(INST1, 1,   1)
`INSTR(INST2, 2,   0)
```

- Include it multiple times:

```
typedef enum {
`define INSTR(name,iop,alu) \
    OP_``name = iop,
`include "ops.vh"
`undef INSTR
} Op;
// Similar for the functions
```

`` is token concatenation

`Agenda

- Standards & Compliance
- Best Practices
- Applications
- Metaprogramming
- Preprocess for Free
- Conclusions
- Q&A

`warning

This is the
`evil
part of the presentation

These examples rely on unspecified language corners, and break several simulators.
See the paper before using!

`Repetition

- Can we implement `REPEAT?

```
`REPEAT(3,d) // -> d d d
```

- Yes, use a lookup table with name generation

```
`define REPEAT(n,d)  `_REPEAT_``n(d)
`define _REPEAT_0(d)
`define _REPEAT_1(d) d
`define _REPEAT_2(d) `_REPEAT_1(d)d
`define _REPEAT_3(d) `_REPEAT_2(d)d
…
```

`Name_Generation

Synopsys Users Group
BOSTON 2010

- Defines can create other defines

```
`DEFINEIT(def,val)  // ~-> `define def val
```

- Allows creating definitions at substitution
- Define names can come from concatenation

```
`define A foo
`define B bar
`define `A```B 1   // `define foobar 1
```

- This enables hash tables!

```
`HASH_SET(key,value)
`HASH_GET(key)
```

`Math

- Hash tables allow mathematical operations

```
`AND(a,b)  // "1" iff a==1 and b==1
`OR(a,b)
`NOT(a)
`LTE(a,b)
`IF_DEFINED(def)
`IF_NOT_DEFINED(def)
```

- This allows most of what's needed for

```
`IF(`OR(A,B))      // Same as C's #if A||B
`FOR(min,max,body)
```

  - But it falls apart in SV2009 – see the paper **sv2012?**

`Agenda

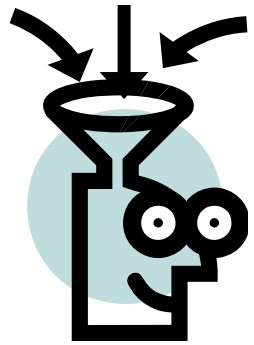
Synopsys Users Group
BOSTON 2010

- Standards & Compliance
- Best Practices
- Applications
- Metaprogramming
- Preprocess for Free
- Conclusions
- Q&A

`vppreproc

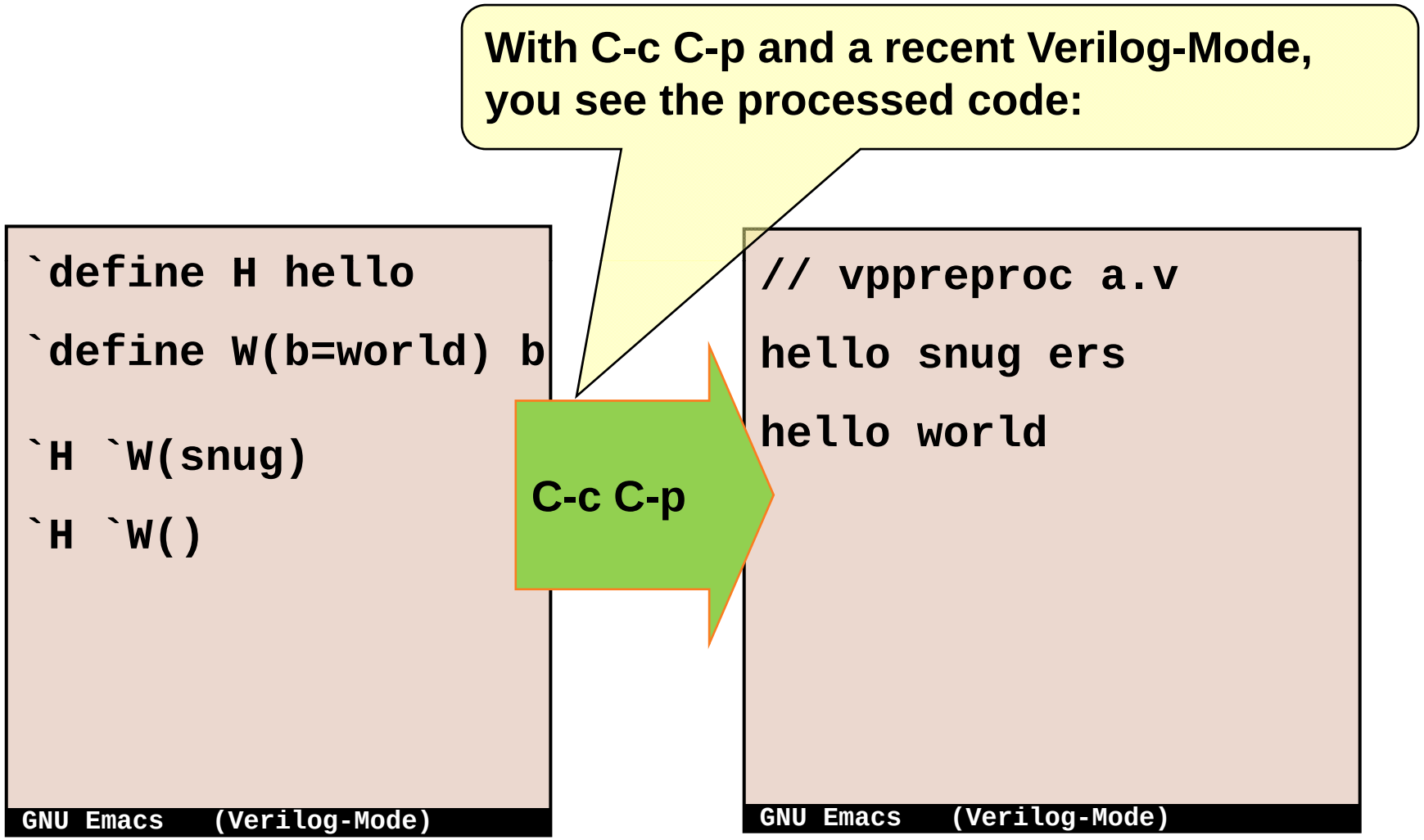
Synopsys Users Group
BOSTON 2010

- How to debug complex macros?
  - VCS macro expansion debug – ask your AE
  - Or, expand the macros to make a new file
  - Similar to “gcc –E”



- vppreproc, part of Verilog-Perl is a full 2009 preprocessor, fully open sourced
  - Output can be fed back into any simulator
  - Also can list all defines, etc.
- Preprocessing is also useful to extend scripts
  - For example add `ifdefs to your TCL scripts
    So the same `defines can feed all tools!

`Emacs_Preprocessor_Debugging



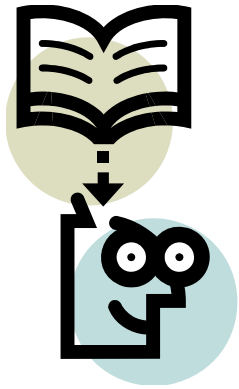
`Agenda

- Standards & Compliance
- Best Practices
- Applications
- Metaprogramming
- Preprocess for Free
- Conclusions
- Q&A

`Conclusions(1)

Synopsys Users Group
BOSTON 2010

- `Good Practices
  - Parenthesize define arguments
  - Use define formals only once to prevent side effects
  - Don’t pass side effects to a macro
  - Wrap “task” macros in do…while
  - Prefix define names to prevent name colisions
  - Undefine local defines at the end of files
  - Use `ifdef, not metacomments



`Conclusions(2)

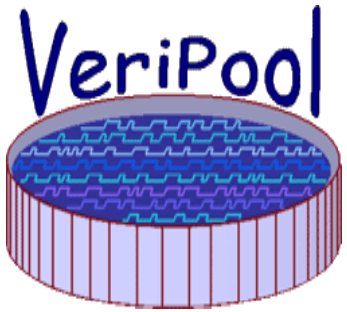
Synopsys Users Group
BOSTON 2010

- Advanced 
  - Defines can create definitions at substitution time
  - The preprocessor can create hash tables
  - Repeat and `if can be emulated
- Debug with vppreproc and Emacs
- Check your simulator compatibility

`Open_Source

Synopsys Users Group
BOSTON 2010

- Open source design tools at http://www.veripool.org
  - These slides + paper at http://www.veripool.org/papers/
  - Verilog-Perl – Toolkit with Preprocessing, Parsing, Renaming, etc
  - Verilog-Mode for Emacs – /*AUTO…*/ Expansion



- Additional Tools
  - rsvn – 10x Faster Subversion from NFS clients
  - SVN::S4 – Subversion with meta-projects, snapshots
  - Verilator – Compile SystemVerilog into C++/SystemC