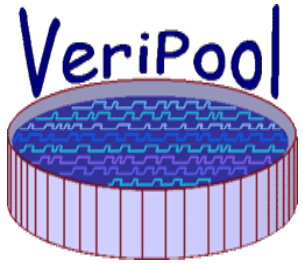


Verilator: *Fast*, Free, But for Me?

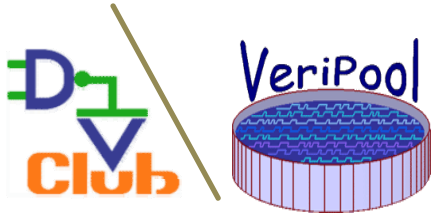
http://www.veripool.org/papers

Wilson Snyder

Cavium Networks

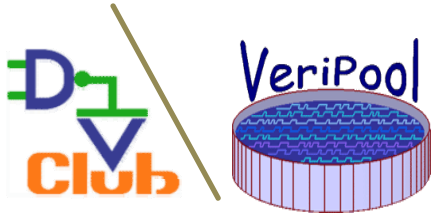
wsnyder@wsnyder.org

Agenda



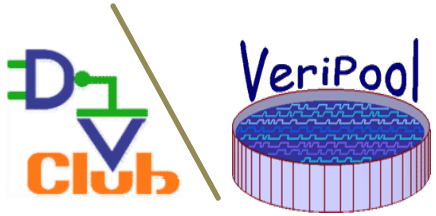
- Why Open Source Simulation?
- Introduction to Verilator
- Getting Started with Verilator
- Verilator Futures
- Other Tools
- Conclusion
- Q & A

Where's Open Source From?



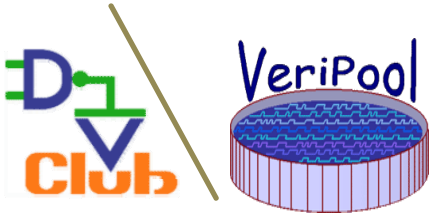
- Generally, not from Hobbyists
- Every company has a pile of hackware, often inefficiently reinventing the same solution
  - Instead they contribute, borrow and improve
- Verilator was of this mode
  - When written, there's was no "Application owns the main-loop" Verilog compiler

Why Write Open Source?



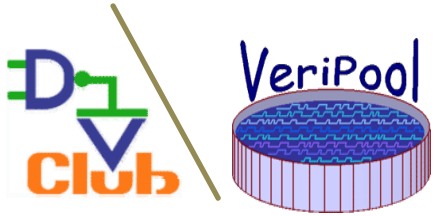
- Authoring open source is often more cost effective than licenses
  - Even if one person spends lots of time (I'm below 10%)
- Contributions by others later benefit employer
  - Certainly more cost effective to share labor with others
  - Other authors wrote many features later needed by my employer
- Much higher documentation quality
- Much higher test quality
- Learn great techniques from other companies

Open Source Advantages (1)



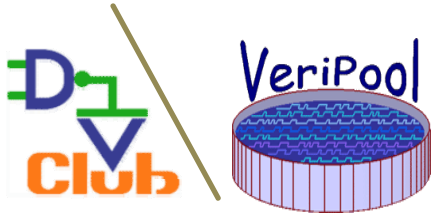
- Financial
  - Cost – iff it does close to what you need
    - Else, need a cost-benefit analysis
      - Not worth rewriting rarely used commercial tools
  - Open License – Required for some applications
    - Example: NXP needed a solution they could provide to software developers, and couldn't contact a license server
    - Example: Running simulations on cloud machines
  - Stronger negotiation position when buying commercial tools

Open Source Advantages (2)



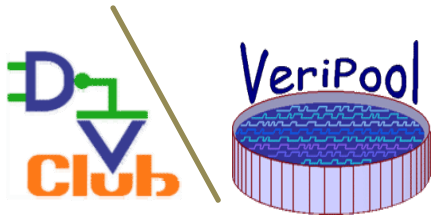
- Source Code Visibility
  - Repurposing
    - Have a similar problem, but need tweaks commercial people are unlikely to want to do
  - Visibility into everyone's bugs, to see what to avoid
    - EDA companies rarely share their bug databases
  - Potentially quick bug turn-around
    - Minutes if you do it yourself!

Open Source Negatives



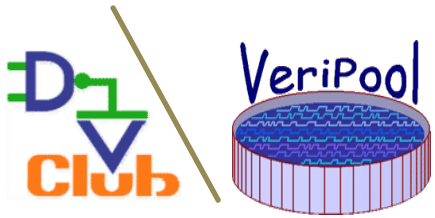
- Support – you're the first level support person
  - There's no guarantees someone else will fix your bug
    - But you could fix it – with old commercial tools, that's often not possible
  - Some open source projects don't take patches back
    - Leads to local versions and hard upgrades
    - Check "liveness" of a project before using their code
  - Few training resource available
- Quality – Varies – as with commercial tools
  - Evaluate as with any other tool
- Features – Often less than commercial tools
  - Never sign off with all eggs in any one simulator

Leverage Both



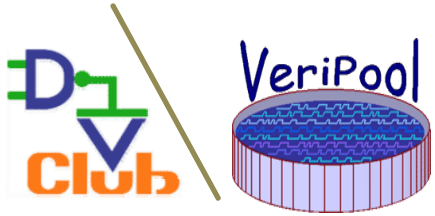
- In summary, from our experiences:
  - Open Source is good for 90% of simulations
  - Commercial is good for 100% of simulations but needed for only 10%
- Don't pay for 9 times more licenses, use both!
- $$ would spend on simulator runtime licenses instead goes to computes
  - ~ 10x more simulations per dollar

Agenda



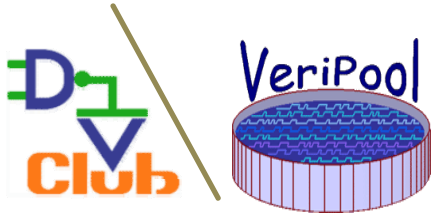
- Why Open Source Simulation?
- Introduction to Verilator
- Getting Started with Verilator
- Verilator Futures
- Other Tools
- Conclusion
- Q & A

History



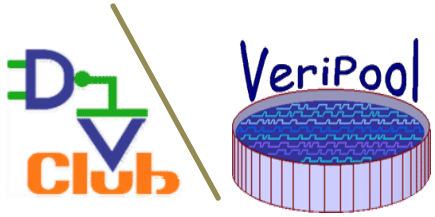
- Verilator was born in 1994
  - Verilog was the new Synthesis Language
  - C++ was the Test-bench Language
  - So Paul Wasson synthesized Verilog into C++
  - And popular open source was, well, GNU Emacs
- Sixteen years later,
  - Three major rewrites
  - Many, many optimizations and language features
  - Much community involvement
  - Open source is proven
    - Who foresaw we would all be using Linux?

Verilator User Base



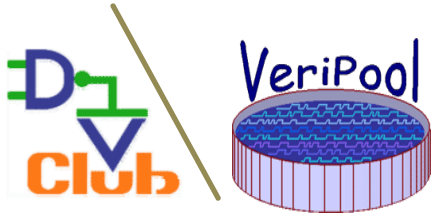
All trademarks registered by respective owners.
Users based on correspondence; there is no official way to determine “users” since there’s no license!

Verilator is a Compiler



- Verilator compiles synthesizable Verilog into C++
  - Matches synthesis rules, not simulation rules
  - Time delays ignored (a <= #{n} b;)
  - Only two state simulation (and tri-state busses)
  - Unknowns are randomized (better than Xs)
- Creates C++/SystemC wrapper
- Creates own internal interconnect
  - Plays several tricks to get good, fast code

Example Translated to C++



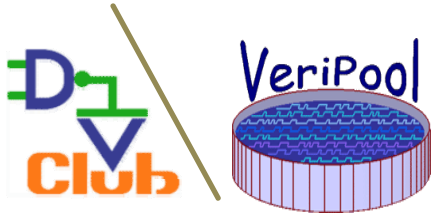
- The top wrapper looks similar to the top Verilog module
- Inputs and outputs map directly to bool, uint32_t, uint64_t, or array of uint32_t's:

```
module Convert;
   input clk
   input [31:0] data;
   output [31:0] out;

   always @ (posedge clk)
      out <= data;
endmodule
```



```
#include "verilated.h"

class Convert {
   bool      clk;
   uint32_t  data;
   uint32_t  out;

   void eval();
}
```

Calling the model

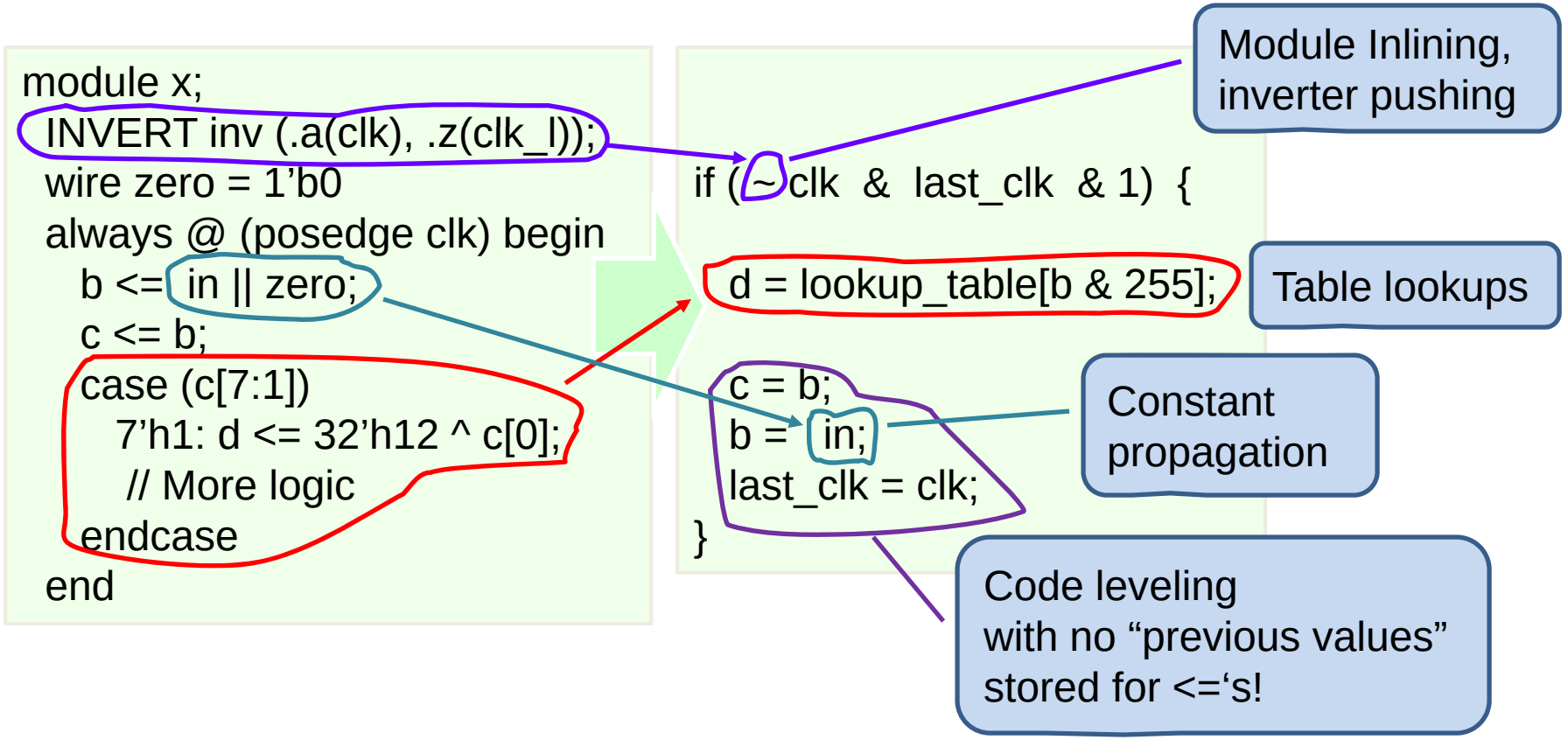


- Application calls the Verilated class in a loop
  - Verilator doesn't make time pass!
    - The key difference from most simulators

```
int main() {
    Convert* top = new Convert();
    while (!Verilated::gotFinish()) {
        top->data = …;
        top->clk = !top->clk;

        top->eval();

        … = top->out();

        time++;  // Advance time…
    }
    top->final();
}
```

```
class Convert {
   bool      clk;
   uint32_t  data;
   uint32_t  out;

   void eval();
}
```

Verilator Optimizations



- End result is extremely fast Verilog simulation

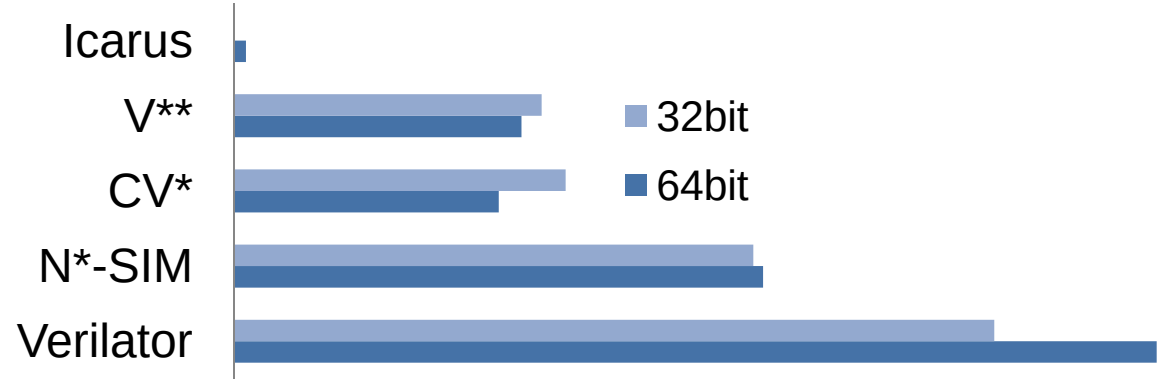
Performance

- Booting Linux on MIPS SoC

- Testbuilder-Based Unit Test

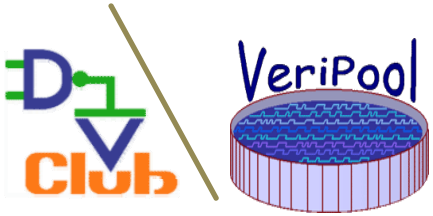
Why so close?
8% in Verilog
92% in C Test Bench
Oh well!

- Motorolla Embedded CPU



As in all benchmarks,
your mileage will vary

Put simply, is Verilator for Me?



- Design in VHDL:
  (Patch wanted ☺)

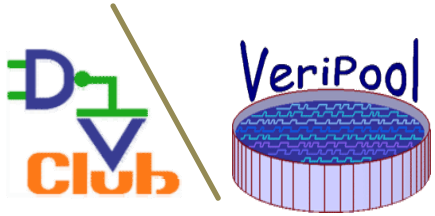
- Design in SystemVerilog,
  big SystemVerilog testbench:

Need full compliance on smaller designs with Verilog2001? Try **Icarus Verilog**

- Design in SystemVerilog,
  Verif/Testbench in C++,
  Limited PLI:

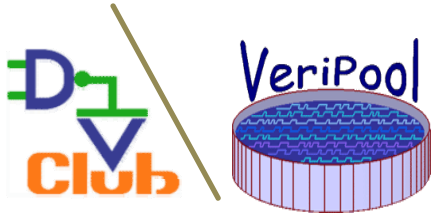
- Design in SystemVerilog,
  testbench in Verilog:

Verilator and Commercial



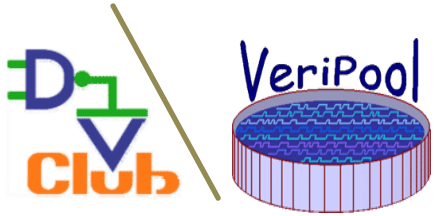
Verilator	Commercial
Synthesizable Verilog-2005 Some SystemVerilog-2009	Mostly SystemVerilog-2005 compliant
C++ and DPI Interface	VPI/DPI interface
Two-State, some tristates	Four-State (0,1,X,Z)
Cycle accurate	Timing accurate (thus required for PLL, PHY and gate simulations)
Limited SVA assertions	Full SVA assertions
Line and Block coverage	Block, FSM, expression coverage
Waveforms, GDB/DDD	Waveforms, source debugger
Faster simulations (1-5x)	Slower simulations
Community support	Excellent customer support
Free	Not quite ☺

Agenda



- Why Open Source Simulation?
- Introduction to Verilator
- Getting Started with Verilator
- Verilator Futures
- Other Tools
- Conclusion
- Q & A

Getting Started

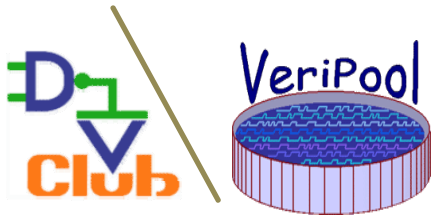


- Download and install
  - Globally: RPMs - Thanks, RPM packagers!
  - or Globally: Download, "**`make install`**"
  - or Cad-tool-ish with multiple versions and env var

    `make ; setenv VERILATOR_ROOT `pwd``
- Follow example in "verilator –help"
- Simple run to see warnings

  `verilator –lint-only –f input.vc top.v`
- Create your own Makefile

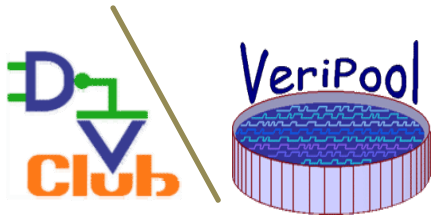
Lint Warnings




```
wire [11:0] foo = in[11:0] + 3'b1;

%Warning-WIDTH: Operator ADD expects  12 bits on
the RHS, but CONST generates 3  bits
Use /*verilator lint_off WIDTH*/ …
```

- Just an advisory - can disable
- Make edits so every vendor's lint tool is happy
  - See "Ten IP Edits Paper": http://www.veripool.org/papers/TenIPEdits_SNUGBos07_paper.pdf

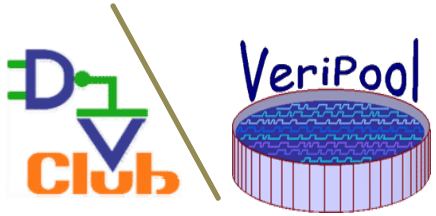
UNOPTFLAT




```
always @*
   a[1] = in
    a[0] = a[1]
%Warning-UNOPTFLAT
```

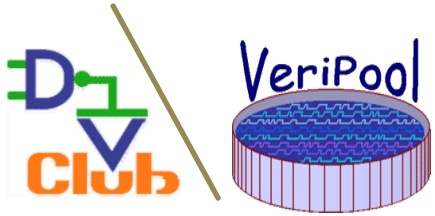
- An always statement(s) will get activated twice on a signal, not one bit change
  - Serious performance problem for Verilator
  - Other simulators also are loosing a little performance
- Split into two always statements
- Rare - one of these for every 100k lines or so
- Wanted: Patch to split bits up for you

Very Large Designs



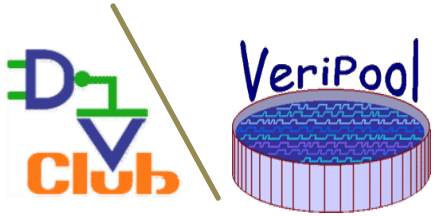
- Verilator is optimized for midsized blocks
- Blocks are assumed to be assembled into chips with C++/SystemC, not Verilog
  - It flattens more than it would if its history was different
  - Patch wanted: Keep hierarchy for specified files
- So, compile time can get large
  - Just like Synthesis, you may need a lot of memory
  - Verilator can split C output
  - Use compile farm with distcc + ccache
  - With these compile time is ~= commercial simulators

Mixing With Other Simulators



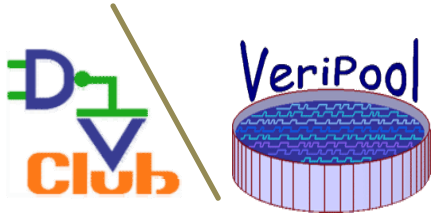
- Run Verilator --lint-only along with your normal lint
  - You may want to turn off other tool's width checks, Verilator's are generally less annoying
- Randomize Xs
  - Finds far more reset bugs than X propagation
  - We can provide PLI code for other simulators
- Use the DPI to connect to all your simulators
  - Much faster than VPI $user calls
  - DPI can't examine the interconnect, though

Debugging Verilated code



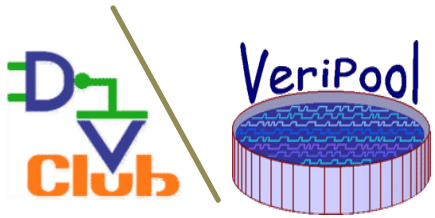
- Sorry.
- Run with –debug
  - Enables internal assertion checks
  - Dumps the internal trees
- Make a standalone test_regress example
  - This will allow us to add it to the suite
  - See the Verilator manpage
- File on Veripool.org bug tracking

Contributing Back



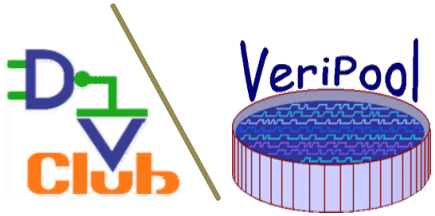
- Use Bug Reporting and Forums
- Try to submit a patch yourself
  - Many problems take only a few hours to resolve yourself; often less time than packaging up a test case for an EDA company!
- Run oprofile and post your bottlenecks
  - Most optimizations came from “oh, it could do better”
- Tell what changes you’d like to see
  - We often have no idea what users find frustrating
- Advocate

Agenda



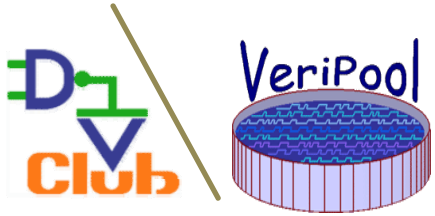
- Why Open Source Simulation?
- Introduction to Verilator
- Getting Started with Verilator
- Verilator Futures
- Other Tools
- Conclusion
- Q & A

Future Language Support



- SystemVerilog Interfaces
  - Patch available – to be integrated
- Structs, Classes
  - Patch wanted – some work started
- Support PLL and DLL models
  - “real” types
  - “time” and timescales
  - New Event Loop
  - Lots of patches wanted; good little projects!

Future Performance



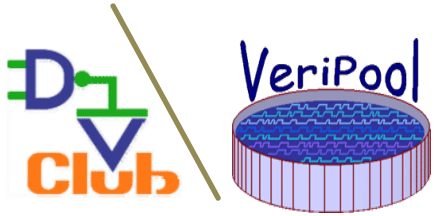
- Avoid replicating large structures
- Eliminate duplicate logic

```
wire a  = b|c;
wire a2 = b|c;
```

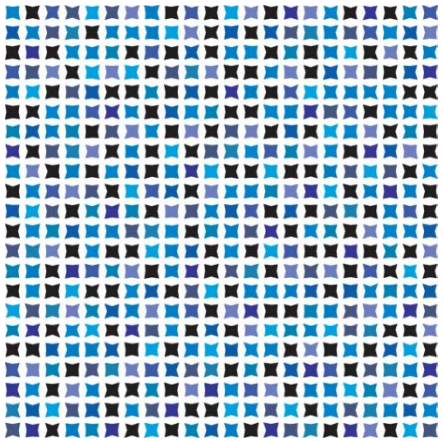
- Optimize Caches
  - Most models are load/store limited
  - On large designs, smaller code footprint with more instructions executed would be faster!

TIP: Buy CPUs with the largest caches you can get, they are generally well worth the premium for ALL simulators.

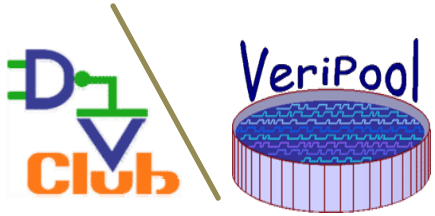
Future Performance



- Multithreaded execution & GPUs
  - Multithreaded/multicore CPUs
  - Commercial sims report up to 7x improvements
    - It's easier when you start from a lower point ☺
  - Hard to avoid communication bottlenecks
  - GPUs – though not great at integer code
  - Great PhD thesis

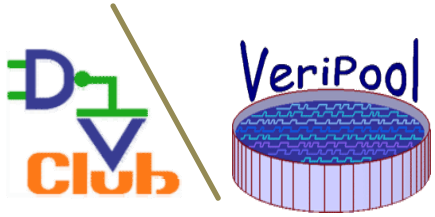


Agenda



- Why Open Source Simulation?
- Introduction to Verilator
- Getting Started with Verilator
- Verilator Futures
- Other Tools
- Conclusion
- Q & A

Verilog-Mode for Emacs



- Thousands of users, including most IP houses
- Fewer lines of code to edit means fewer bugs
- Indents code correctly, too
- Not a preprocessor,
  code is always “valid” Verilog
- ★ Automatically injectable
  into older code.

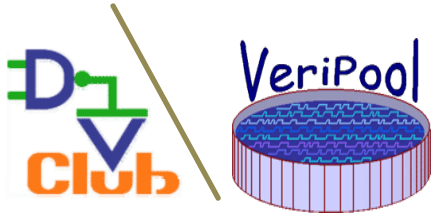
```
  …
/*AUTOWIRE*/

a a (/*AUTOINST*/);
```

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

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

GNU Emacs (Verilog-Mode)

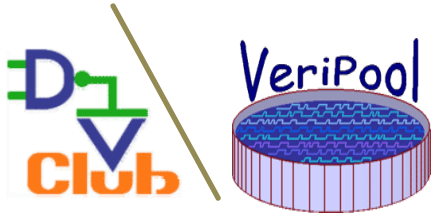
Verilog-Perl Toolbox



- Code shared with Verilator
  - Nearly identical preprocessor
  - Superset of lexical analysis and parser
  - Parses 95% of SystemVerilog 2009
- Vhier
  - Print design hierarchy, input files, etc
- Vppreproc
  - Complete 2009 preprocessor
- Vrename
  - Rename and xref signals across many files

# To	From	Filenames
“a_new”	“a”	“MyMod.v”
“b”	“b”	“MyMod.v”

Verilog-Perl: vpassert



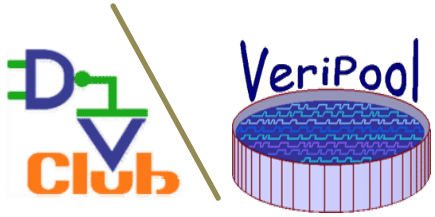
- Preprocessor for messaging, SVA and coverage

```
always @* begin
  if (...) begin
    $ucover_clk(clock, label)
```

- vpassert expands this to:

```
reg _temp;
label: cover property (@(posedge clock) _temp)
always @* begin
  _tempsig = 0;
  if (...) begin
    _tempsig = 1;
```

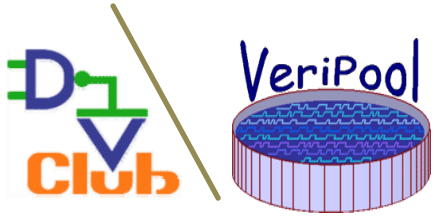
Voneline



- Hard to “grep” for instances in gate level netlists
- Need consistent whitespace
  - Newline only after each cell
  - Preserves // comments and defines
- voneline is a simple filter to accomplish this
  - http://www.veripool.org/voneline

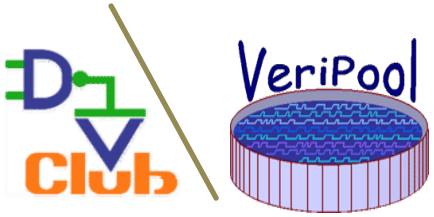
```
module mod;
  input a;
  input b;
  cell1 cell1 (.a(a), .b(b), .c(c), …);
  cell2 cell2 (.x(x), .y(y), …);
endmodule
```

Dir::Project



- Run scripts from a checkout
  - Finds the “root” of a checkout
  - Running “foo” in the shell will find “foo” program in the checkout
  - No need to change PATH
  - Users never “change” projects, current directory controls all

CovVise – Coverage Database



- Uses distributed database, not files
- Tested to > 10 billion bin-inserts per day
  - Above most commercial tools!
- Tracks failing tests and low coverage bins too
- Imports from Verilator coverage

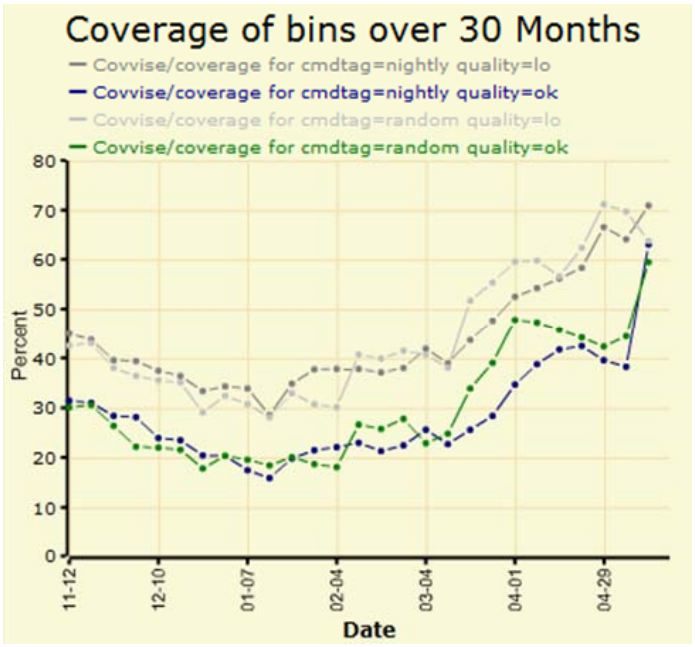
Page tree under Ensemble 6Lycug

Cpu (66.1%) 54.7% 107,379 bins
Ebox (89.6%) 76.4% 1,052 bins
Fbox (40.5%) 19.3% 56,256 bins
cc_bits (93.7%) 5.5% 144 bins

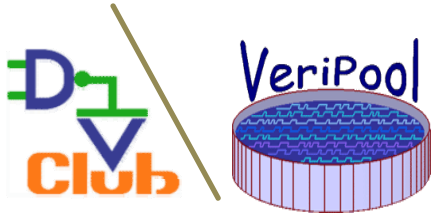
Page: sp_group/Cpu/Fbox/cc_bits
Source: CpuFPredictor.sp 433

Cross of consumers of cc-bits

consumer	relevant_cc_bit dly3							
	cc0	cc1	cc2	cc3	cc4	cc5	cc6	cc7
MOVF_D	1	5	4	3	2	3	2	3
MOVF_PS	3	4	0	0	3	1	4	11
MOVF_S	4	4	30	5	0	6	4	7
MOVT_D	2	fails	2	4	2	3	2	1

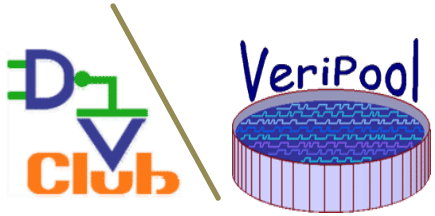


Agenda



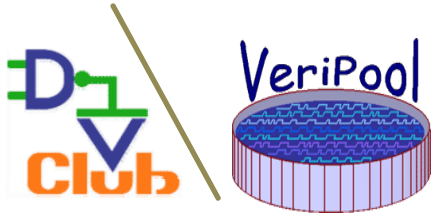
- Why Open Source Simulation?
- Introduction to Verilator
- Getting Started with Verilator
- Verilator Futures
- Other Tools
- Conclusion
- Q & A

Conclusions



- Leverage Open Source AND Commercial Simulators
  - Open Source Simulators
    - Easy to run on laptops or SW developer machines
    - Run as fast as major simulators
  - Commercial Simulators
    - Run analog models, gate SDF delay models, etc
    - Reference for signoff
  - $$ we would spend on 90% of simulator runtime licenses goes instead to computes
    - 10x the throughput!

Sources



- Open source design tools at http://www.veripool.org
  - Downloads
  - Bug Reporting
  - User Forums
  - News & Mailing Lists
  - These slides at http://www.veripool.org/papers/

- Many other tools as described on earlier slides
  - More complete list in the online version of this presentation