Open source SystemVerilog / UVM support and scaling for large designs in Verilator 5.0 and beyond

CHIPS ALLIANCE FALL TECHNOLOGY UPDATE
Sunnyvale, CA, 2022-12-15

Michael Gielda, mgielda@antmicro.com

antmicro

OPEN SOURCE VERIFICATION

- Antmicro is involved with new verification paradigms like Cocotb, but also works with lots of customers who are aligned on UVM
- UVM support in open source tooling was seen as an useful but moonshot endeavour
- Verilator is one of the few widely-accepted open source tools in the industry which can perform better than state-of-the art proprietary software
- But originally, since it was not event-driven, Verilator wasn't even considered capable of handling UVM
- The workload was universally considered to be enormous (mostly because we believe the goals were set in a binary fashion, not incrementally)

RISC-V WEEK IN PARIS

- At May 2022 RISC-V Week in Paris, I presented: *Unlocking open source RISC-V SoC verification*
- Back then, event driven scheduling in Verilator was already implemented, but not mainlined
- Many other issues were identified, but were yet to be implemented; we know however we'd be working on them
- So, how did that go?

SPRING 2022 RISC-V WEEK PARIS
RISC-V
TUESDAY 3RD TO THURSDAY 5TH MAY 2022

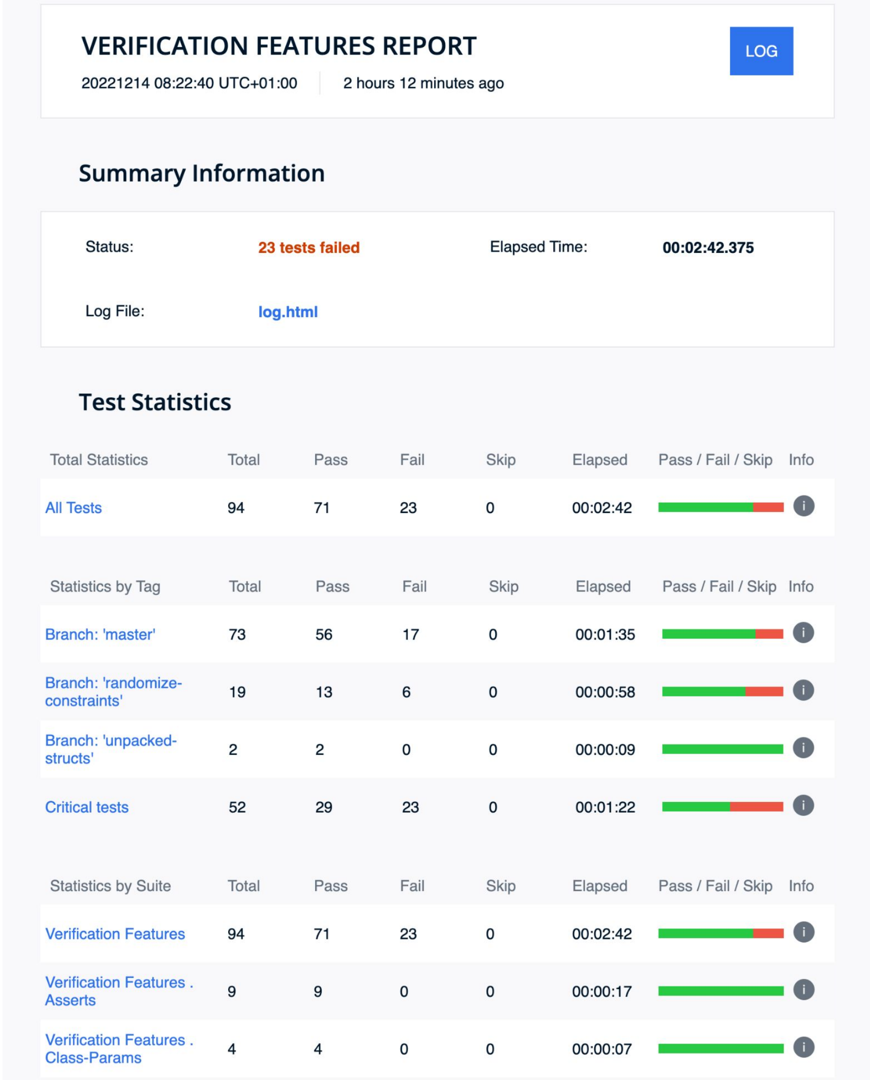
FROM PARIS TO SUNNYVALE

MAINLINING EVENT-DRIVEN SIMULATIONS

- The event-driven simulation capability is now part of Verilator 5.0, which opens the the road to mainline UVM support
- Shout out to Geza Lore and Wilson Snyder with whom we have been collaborating heavily to get this and other things merged
- But this was just the beginning: UVM support cannot be considered as a single thing that has to be "added":
    - Rather, many groups of SystemVerilog features used by the UVM library & specific testbenches

TRACKING PROGRESS

- To track this project, we created a custom test suite and dashboard
  - The dashboard is a convenient way of tracking the progress and catching regressions
  - We focused on making "red" tests pass, not adding more "green" tests - so the absolute numbers of tests does not strictly/proportionally reflect the status
- Most of the code we developed during the project has already been merged into mainline Verilator and is very well maintainable
- Let's see what was done!



VERIFICATION FEATURES REPORT

20221214 08:22:40 UTC+01:00 | 2 hours 12 minutes ago

LOG

Summary Information

Status:	23 tests failed	Elapsed Time:	00:02:42.375
Log File:	log.html		

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip	Info
All Tests	94	71	23	0	00:02:42	🟩🟥	i

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip	Info
Branch: 'master'	73	56	17	0	00:01:35	🟩🟥	i
Branch: 'randomize-constraints'	19	13	6	0	00:00:58	🟩🟥	i
Branch: 'unpacked-structs'	2	2	0	0	00:00:09	🟩	i
Critical tests	52	29	23	0	00:01:22	🟩🟥	i

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip	Info
Verification Features	94	71	23	0	00:02:42	🟩🟥	i
Verification Features . Asserts	9	9	0	0	00:00:17	🟩	i
Verification Features . Class-Params	4	4	0	0	00:00:07	🟩	i

CONCURRENT ASSERTIONS

The support for concurrent assertions in Verilator was very limited.

In addition to assertions themselves, we added value sampling; support for the following keywords has been added/fixed (PR 3569):

- `$rose`
- `$fell`
- `$changed`
- `$stable`
- `$past`

```
 always @(posedge clk) begin
        cyc <= cyc + 1;
        val = ~val;
        $display("t=%0t   cyc=%0d   val=%b", $time,
cyc, val);
    end

assert property(@(posedge clk) cyc % 2 == 1 |=>
(not $rose(val)))
     else $display(" ‘not’ assert failed");
```

CONCURRENT ASSERTIONS CONT.

In addition to sampling improvements, the following features have been added:

- Support for the “not” keyword in assertions (PR 3572)
- Support for named properties in assertions (PR 3667)

```
property check(int cyc_mod_2, logic expected) ;
    @(posedge clk)
      cyc % 2 == cyc_mod_2 |=> val == expected;
endproperty

property check if 1(int cyc_mod_2);
    @(negedge clk)
      check(cyc_mod_2,  1);
endpropert y
```

NEW TYPES ADDED TO VERILATOR

For designs and testbenches we are working with, there are specific signal types that were not supported by Verilator.

For example, wildcard associative arrays (using wildcard index) are now supported: PR 3501.

```
int array [*] = '{default:255};
```

IMPROVED $TEST$PLUSARGS

Verilator did not support expressions (e.g. arbitrary string concatenation) in the arguments of the `$test$plusargs` statement.

Support for this has been added in PR 3489.

```
string bar = "bar";
$test$plusargs({"foo",bar,"baz"});
```

NEW TYPES ADDED TO VERILATOR - UNPACKED STRUCTS

Another important piece are the unpacked structs. Verilator supported only packed structs.

Besides the different memory layout of such structures, those can contain various types inside, which packed structs cannot, e.g. strings. This has been added in PR 3802.

```
typedef struct {
    logic dir;
    logic [7:0] addr;
 } complex_t;

 typedef struct {
    string fst, snd;
 } str_pair_t;
```

antmicro

IMPROVED CLASS SUPPORT

Verilator did have some rudimentary support for classes, however many of the more specific features were not supported.

In UVM class constructors often end with the `return` keyword. In those cases Verilator returned an error due to a missing return type. This was contributed in PR 3734.

Some of the UVM classes contained constructors with arguments. When a child class called this constructor explicitly using `super.new()` , Verilator reported an error. A fix was provided in PR 3789.

```
class foo;
    int x;
    function new(int x);
        this.x = x;
    endfunction
endclass

class bar extends foo;
    function new(int x);
        super.new(x);
    endfunction
endclass
```

VIRTUAL INTERFACES SUPPORT

Verilator did not support virtual interfaces on any level. Those are required by UVM.

Support for virtual interfaces has been added in PR 3654.

```
interface PBus;
   logic req, grant;
   logic [7:0] addr, data;
   modport phy(input addr, ref
data);
endinterface

interface QBus;
endinterface

typedef virtual PBus vpbus_t;

module t (/*AUTOARG*/);

   PBus p8;
   QBus q8;
   vpbus_t v8;
   virtual PBus.phy v8_phy;

(...)

endmodule
```

CLASS TYPE RESOLUTION

UVM makes extensive use of built-in classes provided by the `std` package. Until recently, Verilator didn't allow instantiating classes defined outside the current package. Support for this was added in PR 3755.

There were several outstanding issues in Verilator where `this` keyword used as a reference or function argument was unsupported. Those were fixed by PR 3675.

```
class foo;
endclass

package bar;
    function foo get_foo;
        foo baz = new;
        return baz;
    endfunction
endpackage
```

PARAMETERIZED CLASSES

Parameterized classes are backbone of UVM.
A significant portion of the custom objects definitions in UVM are based on parameterized classes.
Verilator did not support parameters in classes at all.

The following parameterization features have been implemented (PR 3541):

- Class parameters
- Type parameters in classes
- Parameterized subclasses
- Scope resolution when accessing members of parameterized classes

```
class Hold #(int size = 3);
 logic [size-1:0] data;
endclass


class HoldType #(type T = int);
 local T [4:0] data;
endclass

class Holder extends Hold #(7);
endclass;
```

SIGNAL STRENGTH SPECIFIERS

A very basic signal strength support has been added to Verilator. The level of support seems to be satisfactory for the use cases found in a number of tested designs. The following features have been added:

- Statically handling strengths when the strongest signal is a constant: PR 3601
- Resolving assignments with equal strengths: PR 3637
- Tristate signals handling improvements: PR 3604, PR 3629, PR 3551

```
module top (
    input wire clk,
    output wire o
);

   assign (weak0, weak1) o = 0;
   assign (pull0, pull1) o = 1;
   assign (strong0, strong1) o = 1'x;

   always begin
      if (o !== 1)
         $finish;
   end
endmodule
```

BUILT-IN CLASSES USED IN UVM

Verilator lacked support for some of the classes present in the `std` package:

- `std::semaphore`
- `std::mailbox` (parametrized variant)

They are used in several places in the UVM library.

Support for them was added in PR 3708.

```
class mailbox #(T = dynamic_type);
    function new(int bound = 0);
    function int num();
    task put(T message);
    function int try_put(T message);
    task get(ref T message);
    function int try_get(ref T message);
    task peek(ref T message);
    function int try_peek(ref T
message);
endclass
```

LOOP FIXES

Support for the `foreach` loop was limited in Verilator, which led to errors when iterating over a `string` object.

A fix for this was provided in PR 3760.

The `do while` loop was also lacking support for the `break` and `continue` keywords. This required changes to how this kind of loop was processed, which was contributed in PR 3731.

```
function foo;
    string bar;
    foreach (bar[i])
        $display(bar[i]);
endfunction
```

antmicro

SUPPORT FOR `WITH` CLAUSES

Verilator supported the `unique` method, but did not support it with the `with` keyword. This keyword is used for specifying lambda-like constructs. Support was added in PR 3772.

The `with` keyword was also unsupported for queue methods in some contexts:

- when accessing members of complex types,
- when passing queue elements to functions.

Support for those was added in PR 3775 and PR 3739 respectively.

```
class finder;
  typedef struct  {
    string a;
    string path;
  } alias_t;
  protected alias_t m_aliases[$];

  function alias_t  resolve(string
request);
    alias_t results[$];
    results = m_aliases.find(i) with (i.a
== request);
    return results;
  endfunction
endclass
```

CONSTRAINTS

Verilator had rudimentary support for randomization using the `randomize` method. There was no support for constrained randomization - `constraint` declarations were silently ignored.

The testbench defines many complex constraints. To support them, the CRAVE library was extended and heavily adapted to mainline Verilator use as a common frontend for several solvers.

This enabled support for several cases:

- Value-based constraints are supported
- Soft constraints can be added and relaxed
- `with` clauses can be used to define dynamic constraints

```
class Cls;
  rand logic [7:0] val;

  function new;
    val = 0;
  endfunction
endclass

module top ();
  Cls obj;
  initial begin
    int success;
    for (int i = 0; i < 50; i++) begin
      obj= new;
      success = obj.randomize() with { val <
i[7:0] + 5; };
    end
    $finish;
  end
endmodule
```

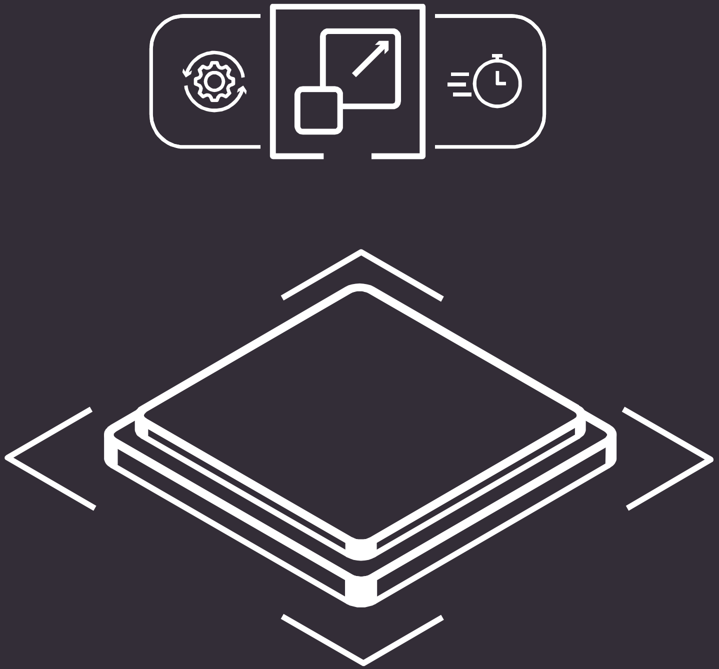
WHAT REMAINS TO BE DONE

- Some of the features that still need to be added to get some basic UVM testbenches to pass:
  - built-in process class
  - `srandom` class method
  - static local variables
  - recursive function/task calls
  - zero delays
  - `constraint_mode` method
  - more complete `randomize(...)` function support
  - more thorough type-parameterized classes support
- And that is just the beginning!
- Still, we are very excited to be where we are

SCALING VERILATOR

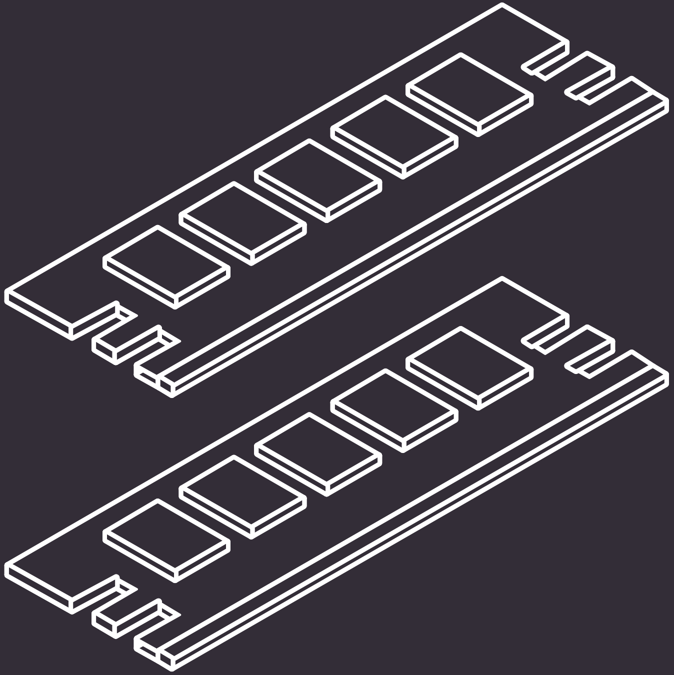
OPTIMIZING FOR SCALABILITY

- An open source UVM capable simulator makes it possible to scale up verification - spawn thousands of machines running massive parallel simulations in CI/CD cloud systems
- Often, the designs we'd like to scale are relatively big - ASIC SoC designs, multicore CPUs, complex communication controllers etc.
- Massive scaling puts greater focus on simulation runtime and memory footprint



REDUCING THE MEMORY FOOTPRINT

- Complex RTL designs generate relatively big verilated models with a lot of internal data structures storing the state of the model
- Recent work on optimizing memory footprint of the simulations focused on:
    - Reducing the internal structures footprint (e.g. removing redundant fields in the internal structs, packing mutually exclusive fields into unions, etc)
    - Verilated model tree object copying reduction
- In total, for relatively big designs (multicore RISC-V SoC class) we have been able to reduce memory usage e.g. **from 72GB to 59G**



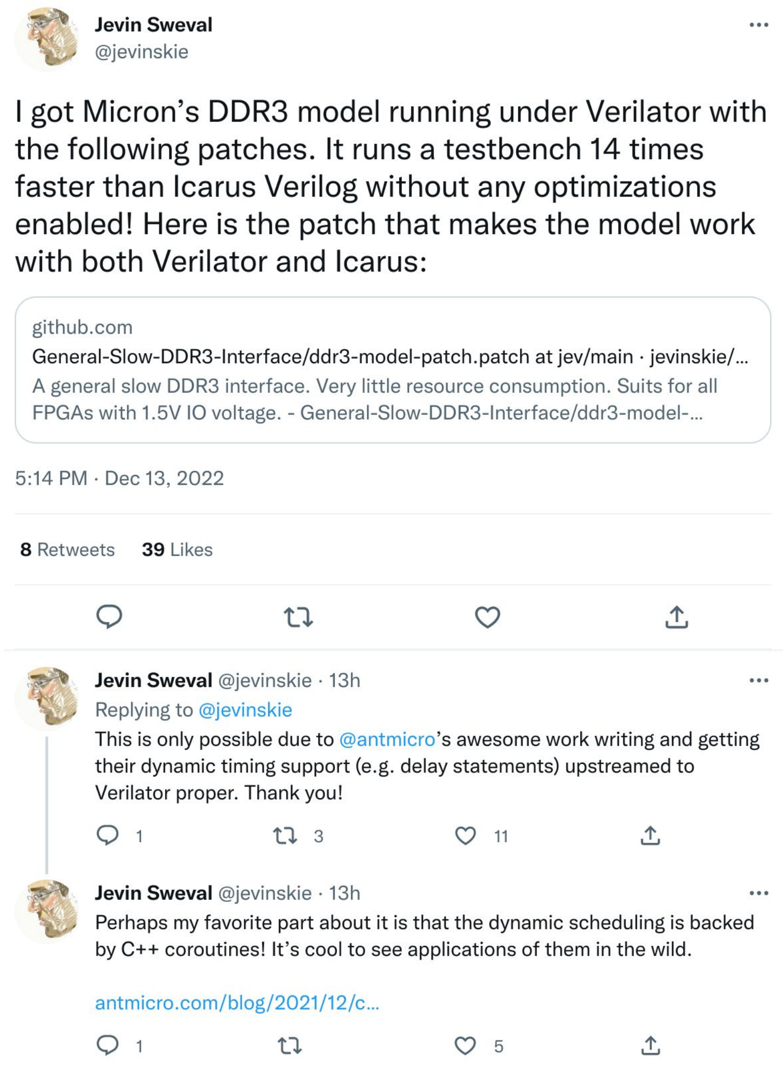
IMPROVING RUNTIME

- Verilator is one of the fastest simulators on the market
- Typically a CI/CD run needs to verilate the design, compile it and run the simulation
- We've introduced a number of improvements to reduce verilation and compilation time
  - Optional disabling of C++ code prettifier
  - Possibility to switch to a faster memory allocator (at a cost of slightly higher memory footprint)
  - Parallel verilation
- For a multicore RISC-V SoC design we've reduced the verilation time from **478s to 249s**

SUMMARY

AS OF TODAY

- Full UVM support in Verilator is still work in progress but we're close to the next significant milestone, running the first UVM testbench
- But even now, our work so far is generating a lot of excitement and practical improvements



Jevin Sweval
@jevinskie

I got Micron's DDR3 model running under Verilator with the following patches. It runs a testbench 14 times faster than Icarus Verilog without any optimizations enabled! Here is the patch that makes the model work with both Verilator and Icarus:

github.com
General-Slow-DDR3-Interface/ddr3-model-patch.patch at jev/main · jevinskie/...
A general slow DDR3 interface. Very little resource consumption. Suits for all FPGAs with 1.5V IO voltage. - General-Slow-DDR3-Interface/ddr3-model-...

5:14 PM · Dec 13, 2022

8 Retweets **39** Likes

Jevin Sweval @jevinskie · 13h
Replying to @jevinskie
This is only possible due to @antmicro's awesome work writing and getting their dynamic timing support (e.g. delay statements) upstreamed to Verilator proper. Thank you!
1 3 11

Jevin Sweval @jevinskie · 13h
Perhaps my favorite part about it is that the dynamic scheduling is backed by C++ coroutines! It's cool to see applications of them in the wild.

antmicro.com/blog/2021/12/c...
1 5

2023 TARGETS

- Get some UVM testbenches running!
- Work with open source projects like Caliptra
  - Open source Root of Trust (RoT) based on VeeR cores
  - Google, AMD, Microsoft, Nvidia
- Enable instruction stream based verification frameworks like RISC-V DV
- If you’d like to speed this up, come talk to us, or email me!

antmicro

THANK YOU
FOR YOUR ATTENTION!

mgielda@antmicro.com