An HTML version of this newsletter can be found at:

http://janick.bergeron.com/guild/1-08.html

(VG 1.08 Content) --- [00/04/14]

The guild archive can be found at http://janick.bergeron.com/guild.
To subscribe, simply send a request to janick@bergeron.com.

(VG 1.08 Editorial) --- [00/04/14]

Thanks to the effort of goodkook [goodkook@nms.anslab.co.kr], the Verification Guild will be partially translated and/or annotated in Korean. As soon as time permits, I'll put the translated versions on the guild's website. And no: I have no plans to personally translate it into French...

I just came back from Verisity's first User's Group meeting (which was called "Club Verification" instead of VUG, because they did not want you to confuse it with Dr Seuss's vug under the rug...).

From Francine Ferguson's welcome address, there were over 100 registered attendees. Unfortunately, only a little more than 40 (my ad hoc count) showed up. Since registration was free, it was to be expected. Several attendees flew in from the east coast, the midwest, Canada, Europe, and Asia. The majority of attendees were experienced Specman users and the presentations, questions, and discussions reflected the verification expertise of the audience. The small size made for high quality and intense exchanges. **My Best Paper Award goes to Heath Chambers for his presentation titled "How to Verify a Switch".**

Verisity seems to want to emulate Synopsys's format and the contingent of R&D engineers was impressive! Many had flown in from Israel for the occasion. It was made very clear that Verisity wants this conference to be user-driven. If you want to participate on the technical committee, send an email to jen@verisity.com.

I hope next year's meeting will include tutorials and more beginner-level presentations, to broaden the audience to include non-Specman users who have an interest in high-level verification languages. It would also likely make for more debates: the uniformity of the audience's background and beliefs on verification sometimes gave the impression of preaching to the choir. Based on the show-of-hand survey in the Vera session at SNUG, where most attendees had no experience with Vera, there is broad interest out there. Maybe its time for "VerifCon", with Vera, Specman, Verilog, VHDL, UML, SDL, and other verification-related technologies under one roof.

- Janick Bergeron
janick@bergeron.com

(VG 1.08, Item 1) -- [00/04/14]

Subject: Looking for recommendations for bug tracking tools

From: Ton van Hekezen [tvhekezen@lucent.com]

We are looking for a "bug track" or "issue track" system for (parts of) our ASIC development. Bugs could by anything from RTL coding bugs, missing library cells and so on. We may also use it to keep of bugs and problems we found in tools.

Does anyone have good experiences with such tools? Public domain like Jitterbug or Bugzilla, commercial like ProjectSync of Synchronicity or ClearCase, whatever. What do you use? What do you like about it and what do miss?

Our requirements of a bug tracking system are:

```
1. Webbased or multi-platform.
    It must be accesible for both Unix users and MS-WinXX users.

2. Common internal format of bugs database.
    ASCII files (or possibility to generate ASCII/HTML/XML files)
    would be preferable. If the software is not useful anymore,
    the data still is.

3. Fields per bug record:
    a. bugnumber
    b. creation date
    c. date of last update
    d. assignee (person who should fix the bug)
    e. status (open, closed, defered)
       A more detailled version of "open" and "closed" could be
        open   =  created, assigned, submitted, tested, deferred
        closed =  approved, killed
       It would be an advantage if this is configurable.
    f. severity (critical, major, minor, wish-list)
    g. abstact (one line)
    h. description (several lines, lines can be added when doing an
       update)
    i. Attachement (any file, possible URL)
    Some configuration would be nice too!

4. Administrator with extra rights.
    A maintainer or administrator should be able to manually delete
    bugs, move bugs, rephrase descriptions, and move the database
    to complete different computers or filesystems.

5. Security.
    Security mechanism to restrict viewing of bug reports, viewing
    bugreports, changing status of bugs etc.
    Furthermore, users should be able to receive a new password
    automatically (i.e. without human intervention) when they forgot
    it.
```

6. It would be an advantage if no external packages (e.g. database)
    must be installed.

7. Open source software is fine, but we also don't mind to
    pay money for good software.

Some usefull starting points for bugtracking software, both open
source and commercial:
   http://www.iac.honeywell.com/Pub/Tech/CM/PMTools.html
   http://alumni.caltech.edu/~dank/gnats.html

(VG 1.08, Item 2, re: VG 1.07, Editorial) -------------------- [00/04/14]

Subject: Attracting engineering talent to verification

> Is having separate verification and design effort the Right Thing to
> do? Why is RTL coding more "fun" than verification? How can you
> attract top engineering talent to the verification task?

----   ----   ----   ----   ----   ----   ----

From: Peet James [peetj@qualis.com]

I found that after 17 years writing RTL (first with propietary
languages, then with VHDL and Verilog), I was getting bored with
entering the same old stuff on new design. The challenge/fun was
typically in the archecting and getting synthesis to close. I got to
'reuse' my code many times, but the synths stuff was so different
(with new DC revs) the one or two times I did it each year, it was
typically a re-learning experience each time.

Then about 6 year ago I started to do more verification. Scary at
first, lots of new problems to solve. Now I like the verification. I
get to reuse a lot of my verification ideas and code, and apply them
on new interfaces. It is challenging and fun in a different way (Major
High Fives when the whole environment compiles or when a big
complicated sim runs).

RTL sort of dead-ended for me, but verification keeps growing and
growing with more layers. I agree with the separate verif team to get
a 2nd set of eyes, and I think once engineers get over the fear of
verification and see how to do it in a cool/fun/structured way, then
they love it.

----   ----   ----   ----   ----   ----   ----

From: Anonymous

I have been READING in various trade magazines for years about proper
ASIC development methodology. 1) Develop an executable specification
in C/C++, 2) Have separate design and verification teams, 3) use OUR
eda tools because they'll make you more productive in BILLION gate
designs... yadayadayada!!!!!!

It all good in print but boy is it difficult to pull-off in the real
world. Verification is just one of many issus that's a pain to do
correctly. Why? Because management often fails to recognize its
importance. My experience has been that engineers typically have all
the responsiblity but virtually no authority when it comes to ASIC
development. Management determines schedules, resources, and people
dedicated to various projects because they control the check book.
And because they have all of the authority they usually fail to staff

for verification properly. They hope to save money by having the
design team also function as the verification team. Now the design
team, operating under a schedule they did not create, are now forced
to cut corners during verification since they're behind schedule!
I've seen this repeated over and over again at different companies.

Also, engineers would rather design than verify because having years
of ASIC design experience on your resume will absolute get you better
compensation and employment than if you only had verification
experience. That's because companies (i.e. management) values design
over verification.

I agree with Mr. J.B. that verification is critical to ASIC
development but too bad the guy/gal the conducts my annual review does
not...

---- ---- ---- ---- ---- ---- ----

From: David Madison [madison@transmeta.com]

The bridge on the cover of your book was verified by the designers. :-)
Designers want to prove that their design works.
Verification engineers want to break the design.

(VG 1.08, Item 3, re: VG 1.07, Item 2) --------------------- [00/04/14]

Subject: C vs VHDL for testbenches

> A question that really puzzles me is why would you use C as a
> verification language with FLI when you can do the verification in
> VHDL, a definitely MORE powerful language. The only thing I can think
> of is the visibility in internal signals...

---- ---- ---- ---- ---- ---- ----

From: goodkook [goodkook@nms.anslab.co.kr]

We had designed MPEG compliance audio decoder. designing this highly
arithmetic intensive design, it's divided into sub-blocks, and each of
sub-block is modeled with C by the sub-designer. remembering MPEG is
lossy compression algorithm, the output is depend on the algorithm
that adopted into hardware. output is varias with designed special
"ALU" and hardware, which must be considered the size and power
consumption. this means, we must consider the accurecy of decoded
output, permissible error rate, and designed hardware size. and more,
to verify the design we need very very large amount of test vectors.

Summarizing our problem was,

1. Input data to be decoded is encoded lossy algorithm, that is highly
   arithmetic, DSP.
2. trade-off between hrdware size and permissable error rate
3. very large amount of test vectors, nearly tens of Mbytes of ascii
   formatted file.

with this situation, we don't have another choice except FLI.
Testbench read binary formatted input data through FLI. Did not use
ascii formatted input. this is very faster than accessing ascii
formatted test vector.

Project leader(me) write C model as a "executable spec", distributed
it as compiled shared object that can be used ModelSim's FLI. the
designer wrote testbench that compare his designed hardware and output

of distributed FLI.

(VG 1.08, Item 4, re: VG 1.07, Item 5) --------------------- [00/04/14]

Subject: Maximizing his/her learning productivity

> Their setup stuff gets you going (i doubt *anyone* has written their
> own .cshrc from scratch in the last 10+ years).

----   ----   ----   ----   ----   ----   ----

From: David Madison [madison@transmeta.com]

Apart from getting a chance to brag that I've written all 500+ lines
in my current set of .cshrc files from scratch, I'm very wary of the
above statement. I agree that newbies need to start with other
people's setups, but far too often they leave it at that.

The end result is that you:
1) Don't know how to configure the tool
2) Might be missing out on some very powerful features
3) Aren't hitting the tool from all sides.

So, whether or not you want to be a shell power-user, what about
tools? What about the spec? What about the DUT? Do you just grab
someone else's configuration and work with that? I think this is a
dangerous mindset, particularly for a verification engineer. If you
want to break a hammer, you don't do it by pounding nails, right? You
try something new, and that's how you're going to find bugs.

(VG 1.08, Item 5, re: VG 1.07, Item 6) --------------------- [00/04/14]

Subject: Verilog PLI question

> Attempting to initialize rams, I really don't want to touch the
> netlist in anyway (the verilog netlist came from a converted viewlogic
> schematic). I've been trying to access the rams using PLI using the
> scoping approach.

----   ----   ----   ----   ----   ----   ----

From: Chris Spear [Chris@Spear.net]

Like Roseanne Roseannadanna used to say on Saturday Night Live, "You
ask a lotta questions!" First of all, if you want to learn more about
the PLI, don't read the IEEE spec as it only shows the mechanics of
the PLI, not how to use it. Instead, get a copy of Stu Sutherland's
book, "The Verilog PLI Handbook" which has over 800 pages showing how
to write PLI applications. Use it once and it will pay for itself.

> Question 1b:
> What's the proper way to return values?

Use tf_putp(0, <value>); or acc_set_value and a lot of supporting code
and structures.

> Question 1a & 2:
> How do I search for and write to all memories?

With PLI 1.0 you can only write to memories with the tf routines, but
they only work on arguments that have been passed to the PLI

application. To find memories requires the acc routines. The only
way I can see to do this is to compile and run first with a PLI
application which searches for memories with acc routines, writes out
a snippet of Verilog code calling the $initialize task for each
memory, then recompile, including this new code. After you do this
once, each run can use the $initialize calls from the previous run.
You can even add in sanity checks to search for all memories and make
sure a new one has not crept in since the previous compile.

The PLI 1.0 code to write to Verilog memories is confusing without the
book, but the basic idea is you call tf_nodeinfo which returns a
s_tfnodeinfo structure, then use pointers inside that to find where
the memory is stored. Each memory location is divided into A & B
values to hold the 4-state values. The A & B values are aligned to
4-byte boundaries.

I'm going to stop here before I give away any more secrets of the
Bavarian Illuminati. For actual code, see Stu's book, or look at my
implementation for $fread in http://chris.spear.net/pli/fileio.tar.gz

(VG 1.08, Item 6, re: VG 1.07, Item 1) ---------------------- [00/04/14]

Subject: Coverage holes with line/block coverage

> This is in the coverage of assign statements. Since assign statements
> are always executed, coverage tools can miss coverage holes.

----   ----   ----   ----   ----   ----   ----

From: Shalom Bresticker [shalom@msil.sps.mot.com]

Just for the sake of accuracy, I want to say that Vigyan is not quite
correct.

VeriSure (now called Verification Navigator), for example, for
TransEDA, did not ignore continuous assignments. Rather, it reported
the number of times one of the right-hand side variables changed,
causing the left-hand side to be re-evaluated.

In addition, I believe it treated the conditional operator (?:) as an
if-else statement, reporting how many times the condition was true and
false in branch coverage.

I agree that it is important to understand well what is covered and
what is not covered by each type of code coverage. For that very
reason, it is also important to be accurate.

----   ----   ----   ----   ----   ----   ----

From: Andrew T. Lynch [drew@surefirev.com]

Disclaimer: I work for Verisity, and am prejudiced.

Different vendors implement expression coverage in different ways. In
Surecov, expression coverage is indeed the coverage metric for
statements of this type. Block coverage alone is not sufficient.

Surecov's expression coverage will tell you that the expression
(a | b) has been true, false or both. In addition, if a and b are scalar,
or if you use logical rather than bitwise or, surecov will tell you
what combinations of a and b made the expression true and false.

> Of course, if my RTL was written as:

```
>
>     always @(a or b or d1 or d2) begin
>       if (a | b) d = d1 + d2;
>       else d = d1 - d2;
>     end
>
> a coverage tool would catch it.
```

This is also true. Code (block and arc) coverage is targeted at the
behavioural designer. This is also true of fsm, expression and event
coverage. Toggle coverage will be the tool of choice if you are do a
significant portion of your design at the structural level.

A good coverage tool will allow you to tailor it to match your design
style and areas of interest, and minimize the overhead it incurs.

----   ----   ----   ----   ----   ----   ----

From: Tom Borgstrom [tom@transeda.com]

You've raised a very good point on the coverage of conditional assign
statements; they are indeed missed with simple line coverage.
TransEDA's Verification Navigator accurately interprets conditional
assign statements like the one you described using branch coverage and
condition coverage metrics.

To show you an example of the kind of information a good code coverage
tool can provide, I've included excerpts from a log file of
Verification Navigator 6.0 run with ModelSim 5.2c on a design
containing a conditional assign statement.

Line 34 of the original source file contains the conditional assign
statement. Notice the statement coverage report indicates how many
times the line has been evaluated during simulation but does not
indicate how the assign statement was resolved. This is the kind of
information you would get running with just line coverage.

```
   Statement coverage information ...
   Count      Line #           Text
              1          module digital_monitor(char6, char5, char4,
 char3, char2, char1, char0, data, clk, rst) ;
   ...
              29         wire          hello;
   ...
   398        34         assign hello = (mem_enable | disp_enable)  ? 1 :  0;
   ...
```

Fortunately, we ran this example with branch coverage and condition
coverage enabled so that we can get more detailed diagnostics. Branch
coverage gives a detailed summary of how many times the construct
resolved to true or false. This would be how you could determine if
the assigned signal achieved both values.

```
   Branch coverage information ...
   Count      Line #        Branch text
              34         assign hello = (mem_enable | disp_enable)  ? 1 :  0;
   199                   mem_enable | disp_enable: was true
   198                   mem_enable | disp_enable: was false
   Total number of times construct entered : 397
```

In this example, both sides of the conditional expression have been

traversed. If this were not the case you could use condition coverage
to determine which subconditions were causing the problem.

```
   Condition coverage information ...
   Line # 34: assign hello = (mem_enable | disp_enable)  ? 1 :  0;
   Decision: mem_enable | disp_enable

   Basic subcondition coverage
   183         : mem_enable  1
   214         : mem_enable  0
   16          : disp_enable 1
   381         : disp_enable 0

   Focused Expression coverage
                 Score Term
   Pass          2/2   mem_enable
   Pass          2/2   disp_enable
   Total         4/4   = 100.0%

   Multiple subcondition coverage
   Count       mem_enable disp_enable
   198         0          0
   16          0          1
   183         1          0
   ****0****   1          1
```

(VG 1.08, Item 7, re: VG 1.07, Item 3) ---------------------- [00/04/14]

Subject: Interpreting the output from $showvars

```
 > When trying to debug things with $showvars I get strange outputs.
 >
 > ADDR_10 (testbench) wire = Stx
 >     St0 <- (testbench.ADDR_10): assign pin = (d ? out : 1'bz);
 >     1'hx, x <-> (testbench.dfd): tranif1 (addr[10], tb_addr[10],
 > control);
 >     1'hx, x <-> (testbench.cntr): tranif1 logic(pin_addr[10],
 > tb_pin_addr[10], select);
 >
 > What does it mean  1'hx, x ?
```

          ----   ----   ----   ----   ----   ----   ----

From: Shalom Bresticker [shalom@msil.sps.mot.com]

1'hx, x means that the signal in question (ADDR_10) is driven to x by
the tranif1. 1'hx is the value in hex format, x is the value in
decimal. Try checking the values of the other signals connected to
the tranif1's.

Generally, it's better to avoid using bidirectional primitives because
they significantly slow simulations.

(VG 1.08, Item 8, re: VG 1.07, Item 10) ---------------------- [00/04/14]

Subject: Sample 'e' code for public consumption

 > Writing in "e" you can unintentionally create several simple syntaxes
 > that will cause performance problems.
 >
 > Since a lot of these things are implementation dependant, and since
 > Verisity doesn't always share its implementation with its customers

> it's really hard to know what is good and what is bad.

---- ---- ---- ---- ---- ---- ----

From: Efrat Shneydor [efrat@verisity.com]

The best coding style when a TCM waits for events:

1) If the event is simple event (eg "wait @fifo_full"), then the temporals engine is optimized and knows to put the TCM on halt until the event is emitted. No performance problems, no tricks are required.

2) If the event is a compound event (eg "wait @fifo_full or @fifo_empty"), the temporals engine cannot recognize this combination automatically, hence it will check every cycle if this combination just occured. This can cost in performance.

   2.1) If this "wait" is the first line in the TCM, like in Sean Smith's FIFO code, Akiva's solution can be a good methodology.

```
        on fifo_full  { start monitor_fifo()};
        on fifo_empty { start monitor_fifo()};
```

   2.2) There is a more general solution, for waits that are anywhere in the TCM body. If there is a compound event you use alot in TCMs and TEs, you can define a new event. And, like said before, waiting for "simple" events is optimized and do not cost in performance.

   If, for example, many places in your TCMs you wait for ("@fifo_full or @fifo_empty"), define a new event fifo_full_or_empty, and in the TCMs (and other TEs) refer to this new event:

```
    event fifo_full_or_empty;
    on fifo_full {emit fifo_full_or_empty};
    on fifo_empty {emit fifo_full_or_empty};

    monitor_fifo() @clk is {
        wait @fifo_full_or_empty;
        //....
    };

    expect fifo_full_or_empty => // ....

    etc.
```

3) Usage of sys.any is not recommended, as this is the highest frequency clock.

   In a code that is to be shareable, you can define a clock and use it as the sampling clock of the TCMs and Temporal expressions. Users of this package can extend the definition of this clock to match the clock of their choice.

```
      struct fifo_<Type {
          event clock is cycle @sys.any;

          fifo_full() @clock is {
               //....
          }
```

```
    In a seperate, project specific, module:

           event clock is only rise('my_env/clk');


( Reference Desk ) ---------------------------------------------- [00/04/14]

Harry Foster and Lionel Bening,
"Coding RTL Verilog to Facilitate an Optimum Verification Flow"
     http://www.isdmag.com/Editorial/1999/coverstory9912.html

     Selecting an RTL verfiable subset and simple coding style
     maximizes the performance of simulation, Boolean equivalence, and
     model checking, while enabling an optimal flow through synthesis
     and physical design


Martin Abrahams, TransEDA Ltd, and Stuart Riches, Texas Instruments Ltd
"Optimize ASIC test suites using code-coverage analysis"
     http://www.ednmag.com/ednmag/reg/1998/052198/11df_05.htm


"Focus Report: Design Verification"
     http://www.isdmag.com/EEdesign/focusreport9909.html


"Test Benches in C Speed Verification by Unifying Emulation and Simulation"
     http://www.isdmag.com/Editorial/1999/systemdesign9911.html


"VHDL constructs and methodologies for advanced-design verification "
     http://www.ednmag.com/ednmag/reg/1999/112499/24ms579.htm


Gary Peyrot, Lattice Semiconductor
"Behavioral modeling in VHDL simulation"
     http://www.ednmag.com/ednmag/reg/1999/102899/22ms562.htm


"Synopsys, de Geus Strum a Billion"

http://www.electronicnews.com/enews/Issue/FreeIssues/2000/03062000/0103061f
-1.asp


"Big Three Aim at Verification Target: Synopsys, Cadence roll HDL simulators"

http://www.electronicnews.com/enews/Issue/RegisteredIssues/2000/03062000/24
f-2.asp


"Design Languages at the Crossroads Rosetta eclipsed by C; Java in the wings?"
     http://www.electronicnews.com/enews/Issue/1999/06211999/25smias.asp


"Covering your HDL chip-design bets "
     http://www.ednmag.com/reg/1998/102298/22df1.htm


"0-In Releases Verification IP Products for SOC Design", Press Release
     http://www.0-in.com/subpages/news/index.html


( Networking ) -------------------------------------------------- [00/04/14]
( End ) --------------------------------------------------------- [00/04/14]
```