**System Verilog Requests – UVM WG**

**Categories:**

* Priority
	+ High – expected to be seriously considered by SV committee and prioritized within the range of items to be worked on
	+ Medium – expected to be reviewed, discussed, and prioritized by the SV committee
	+ Low – nice to have enhancement
* Impact to UVM:
	+ Impacts Imp of UVM
	+ Impacts API of UVM
	+ Does not impact UVM

**Feature Requests:**

1. Enhance to include Java-Style weak references & Finalize() – Mantis 4697

Requestors: NVIDIA

* High, no alternatives
* Impacts Imp of UVM
* Impacts mixed language dev
1. Enhance process::await to be able to unblock when status() reaches a specified state (or states).

Requestors: NVIDIA

* High, no alternative, Used for process control and better error checking in BCL.
* Impacts Imp of BCL
1. Enhance to allow function overloading.

Requestors: Freescale

* High, painful alternatives
* Impacts API of UVM
1. Enhance to allow task overloading.

Requestors: Freescale

* High, painful alternatives
* Impacts API of UVM
1. Allow constraint reference at constraint expression. See mantis 2968
Requestors: NVIDIA (originally opened by Mirek Forczek)
* Request for constraints to be first class citizens in the objects
* High, no alternative (significant overhead in BCL to work around this missing feature),
* Impacts Imp & API of UVM
1. Ability to select behavior of “default” bins (cover, ignore, illegal). See mantis 4698.

Requestors:NVIDIA, Cisco

* High, very painful alternative to specify everything
* Does not impact UVM
1. Enhance to allow extensions of covergroups and items. See mantis 4703.
Requestors: Cisco
* AOP for covergroups
* High (for Cisco)
* Does not impact UVM
1. Enhance to allow extensions of enumerated types. See mantis 2991.

Requestors: Freescale, Cisco

* Med, using strings is an alternative, other class based alternatives.
* Impacts Imp of UVM
1. Enhance interfaces and modports to allow for inheritance.

Requestors: Freescale, NVIDIA

* Med, no alternatives just lots of painful workarounds.
* Does not impact UVM
1. Enhance to add support for Exceptions. Mantis 2995 and 4704

Requestors: Freescale, NVIDIA

* Med, Would help how, for example, sequences are caught.
* Impacts Imp of BCL
1. Enhance to add Aspect Oriented Programming features. See mantis 3002.

Requesters:Synapse-DA

* Med, more difficult alternatives.
* Impacts Imp and API of UVM
1. Enhance to add a singly rooted class hierarchy. See mantis 4529.
* All classes inherit from a single super base/root class, anything can be passed to a function. This is the idea of uvm void, properly implemented in SV.
* Med, alternative is uvm void.
* Impact Imp and API of UVM
1. Ability to declare/qualify classes/methods/variables/constraints final. See mantis 3004

Requestors:Intel, NVIDIA

* Med, Alternative keyword ‘protective’ does not work
* Impacts Imp of UVM
1. Ability to declare/qualify methods/variables/constraints initial. See mantis 4699

Requestors:NVIDIA

* Med, Alternative keyword ‘protective’ does not work
* Does not impact UVM
1. enum::from\_name(string s) (or equivalent).

Requestors: NVIDIA

* Low, alternative in place for UVM.
1. Enhance to allow for true multiple inheritance (e.g. like what is in C++). Rob Slater.

Requestors: Freescale, NVIDIA, Cisco

* Low, interface class, closures, mix-ins, traits, impacts BCL (would simplify implementation and memory footprint)
1. Clarify that the following can be called w/o void'() protection, and will still be treated as non-time-consuming methods:
	1. rand\_mode(bit on\_off)
	2. constraint\_mode(bit on\_off)
	3. $cast()

Requestors: Freescale, NVIDIA

* Low, alternative to use void protection, impacts end users
1. (alternatively change the definitions of the above methods to a function which always requires void'() protection) .

Requestors: Freescale, NVIDIA

* Low, alternative to use void protection, impacts end users
1. Enhance to add default constraints. See mantis 2988.

Requesters: Synapse-DA

* Low, can use soft constraints for most cases. Impacts end users
1. Enhance to add an exit status. See mantis 4531.

Requesters:Synapse-DA

* Low, work around in the language DPI terminator “call exit” with status
* Does not impact UVM
1. Allow C99-style intermingled declarations in automatic methods. See mantis 4700.

Requestors: NVIDIA

* Low, alternative is to put the declarations first.
* Does not impact UVM
1. Enhance to allow string forms of enumerated variables.

Requestors: Freescale

* No discussion in the UVM committee
1. Macros - Evaluate Strings before Macro substitution
        In ‘C’, we can write following for example:

#define GET\_REG\_TYPE\_EXPAND(regtype) #regtype
 #define GET\_REG\_TYPE\_STR(regtype) GET\_REG\_TYPE\_EXPAND(regtype)
        #define GET\_REG\_TYPE(fieldtype, T) T = (GET\_REG\_TYPE\_STR(fieldtype) == "u8") ? 1 : (GET\_REG\_TYPE\_STR(fieldtype) == "u16") ? 2 : 4;

        In the GET\_REG\_TYPE\_EXPAND, i could use # before macro argument to evaluate it and get the value. I don't think something similar exists in SV ?

Requesters: Applied Micro.

1. Functional Coverage - One-hot toggle coverage support using a keyword isn't available. There are work-arounds suggested with 2012 standard, like writing encoding array and then telling the bins to cover only the encoding array values, but that's a tedious task. Native support would be helpful.

Requester: Applied Micro.

1. Associative arrays -  .key function for associative arrays is not available. The .find function is cumbersome for simple tasks.
        For example:

       int aa[string];

       aa["a"] = 0;
       aa["b"] = 1;
       aa["c"] = 2;
       aa["d"] = 3;
       aa["e"] = 4;

       //to get the key of value 0 in sv
       begin
          int f[$];
          f = aa.find( item == 0); //f[0] will hold key "a" now
       end

       //to get the key of value 0 in specman
       aa.key(0)

Is it feasible given that associative array can be defined as [\*]?  Could a key function work for all other associative arrays and give an error for [\*]?

Requesters: Applied Micro, Cisco.

1. Specify the format of **$typename()** that can be used in the get\_type\_name() function.  (Or specify another system-function for this purpose.)
	1. The one attempt I had to use this with one vendors tools produced a \***huge**\*string, with all the contents and extensions of all super classes.  This wasn’t usable in this form.
	2. This is a UVM problem, rather than SV, but just defining your own get\_type\_name() doesn’t always work as there’s a static version in the object\_wrapper that sometimes gets used (eg. in the default sequences).
	3. Obviously, once in the SV language, I’d expect that UVM would include this in the base-class library.

Requesters: Altera

1. Specify **type**(**this**) to be valid anywhere in a class so that it can be used in the `uvm\_\*\_param\_utils\* macros.
	1. Currently, this is only valid in tasks, constraints, etc.
	2. This prevents a lot of repeat typing (and associated mistakes) of the parameter list – we have some large parameter lists as constraint knobs so we can easily use factory overrides.
	3. With this and point 1., it should be possible to make the factory registration macros not require any parameters (or at least make them optional).

 Requesters: Altera

1. Specify that a covergroup handle can be compared with **null** to test whether it’s been constructed.  (I couldn’t find this anywhere in the LRM.)
	1. Small point, but prevents needing a separate variable.

 Requesters: Altera

1. Specify some kind of “this” for interfaces, so I can write something like:

interface my\_bfm\_if;

bit init = init\_me();

function bit init\_me()

    string hdl\_path = $sformatf("%m");

    uvm\_resource\_db #(my\_bfm\_if)::set(hdl\_path, "bfm\_if", **this\_if**);

return 1;

endfunction : init\_me

endinterface : my\_bfm\_if

We currently use this kind of scheme (this is simplified – we pass a class which contains the interface handle), but need to have an enclosing wrapper to get a handle on the interface.  We can then build up the hdl\_path in the “envs” and retrieve the interface handle using a string in the resource db (ie. no hard-coded hierarchical references).

<http://www.eda.org/mantis/view.php?id=4300>

Requesters: Altera

1. A more general point, it would be \***really**\* good to somehow reduce the “overhead” code that needs to be written to use UVM, eg:
	1. Mechanisms to implement some of the field macros in the SV base language, or at least to facilitate easier use (eg. not having to declare everything at least twice)
		1. I did propose that we not use the field macros and hard-code the do\_\* functions.  However, people didn’t always write the functions and when they did, they contained bugs (more code, mode bugs…)
		2. Whilst development environments can ease this, the few I’ve used seem to be good at generating “templates” when you start, but then you have to maintain the multiple copies by hand.
		3. Something like the e approach of adding extra attributes at declaration time would be good

I raise this, without a comprehensive list of solutions, as this has been a criticism from several quarters when adopting UVM – you have to write a lot of structures/repetitive code.  I believe this is in part because the approach has been to define a methodology orthogonally to the SV language.  Whilst this is flexible/powerful, it isn’t particularly concise or user-friendly.

Requesters: Altera.