Accellera Universal Verification Methodology (UVM, IEEE 1800.2-2017)

Scope

This kit provides a Systemverilog library matching the requirements of IEEE 1800.2-2017. See details in the Library Release Description below.

Note: The implementation provided deviates from the 1800.2-2017 standard, see [DEVIATIONS.md] for additional details.

Kit version

1800.2-2017 1.0

This kit was generated based upon the following git commit state: 60b56b09e9e78bfec23575529e37fed8d0ccc757.

License

Copyright

Contacts and Support

If you have questions about this implementation and/or its application to verification environments, please visit the Accellera UVM 2017 - Methodology and BCL Forum or contact the Accellera UVM Working Group (uvm-wg@lists.accellera.org).

Installing the kit

Installation of UVM requires first unpacking the kit in a convenient location.

```
% mkdir path/to/convenient/location
% cd path/to/convenient/location
% gunzip -c path/to/UVM/distribution/tar.gz | tar xvf -
```

Follow the installation instructions provided by your tool vendor for using this UVM installation and tool version dependencies.

Prerequisites

- IEEE1800 compliant SV simulator. Please check with your tool vendor for exact tool version requirements.
- C compiler to compile the DPI code (if not otherwise provided by tool vendor)

Library Release description

Each class and method in the standard is annotated in the implementation, allowing tools to identify the corresponding section in the standard.

Example:

```
// @uvm-ieee 1800.2-2017 auto 16.5.3.2
extern virtual function void get_packed_bits (ref bit unsigned stream[]);
```

In addition to the APIs described in the standard, the Library includes the following categories of extra API:

1. APIs that are being considered for contribution to the IEEE by Accellera. They are identified by the following annotation:

   ```
   // @uvm-contrib Potential Contribution to 1800.2
   ```

2. APIs that are not being considered for contribution to the IEEE. Generally these are provided for debug purposes. They are identified by the following annotation:

   ```
   // @uvm-accellera Accellera Implementation-specific API
   ```

3. Deprecated UVM 1.2 API
   Note: The deprecated UVM 1.2 APIs are under a `` `ifdef UVM_ENABLE_DEPRECATED_API `` guard. These APIs are only supported when the UVM 1.2 API didn't contradict 1800.2-2017 API. When `UVM_ENABLE_DEPRECATED_API` is defined both the 1.2 and 1800.2 APIs are available. When

`UVM_ENABLE_DEPRECATED_API` is *not* defined, the UVM 1.2 APIs are not available, and any code referencing them will miscompile.

These APIs will only be supported until the next release of the 1800.2 standard. Code leveraging these UVM 1.2 APIs should be migrated to 1800.2 standard APIs to maintain compatibility with future versions of the implementation.

By default, `UVM_ENABLE_DEPRECATED_API` is not defined.

4. APIs used within the library that are not intended to be directly used outside of the implementation.

Note: While the Accellera UVM Working Group supports the APIs described in (1), (2) and (3) above, these APIs are technically not a part of the 1800.2 standard. As such, any code which leverages these APIs may not be portable to alternative 1800.2 implementations.

Backwards Compatibility Concerns

These are instances wherein the functionality of an API that exists in both UVM 1.2 and the IEEE 1800.2 standard has changed in a non backwards-compatible manner.

1. Mantis 6644 Changes to big endian support in uvm_reg_map. In previous versions of UVM, configuring a uvm_reg_map to UVM_BIG_ENDIAN didn't change the byte ordering for data (effectively always forcing LITTLE endian, regardless of configuration). This has been corrected in the 1800.2 release, such that the data provided to the adapter is now properly ordered. Users can update their adapter to handle the new properly ordered data, or configure their map to UVM_LITTLE_ENDIAN.
2. Mantis 6773 Changes to physical address calculation in uvm_reg_map. In previous versions of UVM, address calculation for memory objects narrower than the enclosing map was done as unpacked addressing. This functionality has changed in this release of the UVM library to packed addressing. This implies that, for a memory object whose width is less than the enclosing map width, address of the Nth word of the memory ($N > 0$) will be different than what was calculated in previous versions of UVM. For future releases of the library, API changes to the register layer will be required to support both packed and unpacked modes of physical address calculation.
3. Changes to physical address calculation in uvm_reg_map require that

memory width be an integer multiple of 8 bits(k*8).

4. Changes to physical address calculation in uvm_reg_map require that, where the map and memory widths do not match, the larger width be an integer multiple of the smaller width (larger_width = k*smaller_width).

5. As of 1800.2, the uvm_packer class no longer contains a knob for "big_endian". While the big_endian knob *is* included when running with deprecated APIs enabled, the default polarity of the bit is inverted. This ensures that the packer will operate the same by default regardless of whether deprecated APIs were present. Users relying on the old behavior will need to explicitly set big_endian to '1'.

Migration instructions

In order to migrate to the Library version of the release, It is recommended that you perform the following steps to get your code to run with this release of UVM.

1. Compile/Run using a UVM1.2 library with `UVM_NO_DEPRECATED` defined. This will ensure that your code runs with UVM 1.2 which was a baseline for the IEEE 1800.2 standard development.

Note: All code deprecated in UVM 1.2 has been removed from this version of the library.

2. Compile/Run using this library with `UVM_ENABLE_DEPRECATED_API` defined. This step helps identify the areas where your code may need modifications to comply with the standard.

3. Compile/Run using this library without `UVM_ENABLE_DEPRECATED_API` defined. Removing the define ensures that only the 1800.2 API documented in the standard, along with any non-deprecation accellera supplied API, is used. Any new compile failures are the result of deprecated 1.2 APIs.