**Avalon-MM VVC** –Quick Reference

**VVC**

For general information see UVVM VVC Framework Essential Mechanisms located in uvvm\_vvc\_framework/doc. **CAUTION**: shaded code/description is preliminary

|  |
| --- |
| avalon\_mm\_write (VVCT, vvc\_instance\_idx, addr, data, [byte\_enable,] msg, [scope]) |
| Example: avalon\_mm\_write(AVALON\_MM\_VVCT, 1, x"00006000", x”AABBF102”, “Writing to Peripheral 1”); |

*avalon\_mm\_vvc.vhd*

|  |
| --- |
| avalon\_mm\_read (VVCT, vvc\_instance\_idx, addr, [TO\_SB,] msg, [scope]) |
| Example: avalon\_mm\_read(AVALON\_MM\_VVCT, 1, x"10056000", “Reading from Peripheral 1”);  avalon\_mm\_read(AVALON\_MM\_VVCT, 2, x"1005F000", TO\_SB, “Read from Peripheral 2 and send to Scoreboard”); |
|  |

|  |
| --- |
| avalon\_mm\_check (VVCT, vvc\_instance\_idx, addr, data, msg, [alert\_level, [scope]]) |
| Example: avalon\_mm\_check(AVALON\_MM\_VVCT, 1, x"FF113000", x”0000393B”, “Check data from Peripheral 1”); |

|  |
| --- |
| avalon\_mm\_reset (VVCT, vvc\_instance\_idx, num\_rst\_cycles, msg, [scope]) |
| Example: avalon\_mm\_reset(AVALON\_MM\_VVCT, 1, 5, “Resetting Avalon-MM interface for 5 cycles”); |

|  |  |  |
| --- | --- | --- |
| avalon\_mm\_lock (VVCT, vvc\_instance\_idx, msg, [scope]) |  | avalon\_mm\_unlock (VVCT, vvc\_instance\_idx, msg, [scope]) |
| Example: avalon\_mm\_lock(AVALON\_MM\_VVCT, 1, “Locking Avalon MM Interface”); |  | **Example**: avalon\_mm\_unlock(AVALON\_MM\_VVCT, 1, “Unlocking Avalon MM Interface”); |

Avalon-MM VVC Configuration record **´vvc\_config´ -** accessiblevia **shared\_avalon\_mm\_vvc\_config**

**Common VVC procedures applicable for this VVC**  
- See UVVM Methods QuickRef for details.

**await\_completion**() **enable\_log\_msg**() **disable\_log\_msg**()

**fetch\_result**()

**flush\_command\_queue**()  
**terminate\_current\_command**() **terminate\_all\_commands**() **insert\_delay**()

**get\_last\_received\_cmd\_idx()**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **C\_AVALON\_MM\_VVC\_CONFIG\_DEFAULT** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AVALON\_MM\_INTER\_BFM\_DELAY\_DEFAULT |
| [cmd/result]\_queue\_count\_max | natural | C\_[CMD/RESULT]\_QUEUE\_COUNT\_MAX |
| [cmd/result]\_queue\_count\_threshold | natural | C\_[CMD/RESULT]\_QUEUE\_COUNT\_THRESHOLD |
| [cmd/result]\_queue\_count\_threshold\_severity | t\_alert\_level | C\_[CMD/RESULT]\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY |
| use\_read\_pipeline | boolean | true |
| num\_pipeline\_stages | natural | 5 |
| bfm\_config | t\_avalon\_mm\_bfm\_config | C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT |

Avalon-MM VVC Status record signal **´vvc\_status´ -** accessiblevia **shared\_avalon\_mm\_vvc\_status**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** |  |
| current\_cmd\_idx | natural |  | |
| previous\_cmd\_idx | natural |  |
| pending\_cmd\_cnt | natural |  |

VVC target parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| VVCT | t\_vvc\_target\_record | AVALON\_MM\_VVCT | VVC target type compiled into each VVC in order to differentiate between VVCs. |
| vvc\_instance\_idx | integer | 1 | Instance number of the VVC |

VVC functional parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| addr | unsigned | x”0000325A” | The address of a Avalon-MM accessible register. Could be offset or full address depending on the DUT |
| data | std\_logic\_vector | x”F1A332D3” | The data to be written (in avalon\_mm\_write) or the expected data (in avalon\_mm\_check). |
| byte\_enable | std\_logic\_vector | (others => ‘1’) | This argument selects which bytes to use (all ‘1’ means all bytes are updated) |
| msg | string | “Send to peripheral 1” | A custom message to be appended in the log/alert |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the method. |
| scope | string | “AVALON MM VVC” | A string describing the scope from which the log/alert originates. In a simple single sequencer typically  "AVALON MM BFM". In a verification component typically "AVALON MM VVC ". |

VVC entity signals

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| clk | std\_logic | VVC Clock signal |
| avalon\_mm\_vvc\_master\_if | t\_avalon\_mm\_if | See Avalon-MM BFM documentation |

VVC entity generic constants

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Default** | | **Description** | |
| GC\_ADDR\_WIDTH | integer | 8 | | Width of the Avalon-MM address bus | |
| GC\_DATA\_WIDTH | integer | 32 | | Width of the Avalon-MM data bus | |
| GC\_INSTANCE\_IDX | natural | 1 | | Instance number to assign the VVC | |
| GC\_AVALON\_MM\_CONFIG | t\_avalon\_mm\_bfm\_config | C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT | | Configuration for the Avalon-MM BFM, see Avalon-MM BFM documentation. | |
| GC\_CMD\_QUEUE\_COUNT\_MAX | natural | 1000 | | Absolute maximum number of commands in the VVC command queue | |
| GC\_CMD\_QUEUE\_COUNT\_THRESHOLD | natural | 950 | | An alert will be generated when reaching this threshold to indicate that the command queue is almost full. The queue will still accept new commands until it reaches C\_CMD\_QUEUE\_COUNT\_MAX. | |
| GC\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | t\_alert\_level | WARNING | | Alert severity which will be used when command queue reaches GC\_CMD\_QUEUE\_COUNT\_THRESHOLD. | |
| GC\_RESULT\_QUEUE\_COUNT\_MAX | natural | 1000 | Maximum number of unfetched results before result\_queue is full. | |
| GC\_RESULT\_QUEUE\_COUNT\_THRESHOLD | natural | 950 | An alert with severity 'result\_queue\_count\_threshold\_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0. | |
| GC\_RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | t\_alert\_level | WARNING | Severity of alert to be initiated if exceeding result\_queue\_count\_threshold | |
|  |  |  | |  | |

VVC details

All VVC procedures are defined in vvc\_methods\_pkg (dedicated this VVC), and uvvm\_vvc\_framework.td\_vvc\_framework\_common\_methods\_pkg (common VVC procedures)

It is also possible to send a multicast to all instances of a VVC with ALL\_INSTANCES as parameter for vvc\_instance\_idx.

*Note: Every procedure here can be called without the optional parameters enclosed in [ ].*

# VVC procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **avalon\_mm\_write()** | **avalon\_mm\_write(VVCT, vvc\_instance\_idx, addr, data, [byte\_enable,] msg, [scope])**  The avalon\_mm\_write() VVC procedure adds a write command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the write command is scheduled to run, the executor calls the Avalon-MM BFM avalon\_mm\_write() procedure, described in the Avalon-MM BFM QuickRef.  avalon\_mm\_write can be called with or without byte\_enable constant. When not set, byte\_enable is interpreted as all ‘1’, indicating that all bytes are valid.  Examples:  avalon\_mm\_write(AVALON\_MM\_VVCT, 1, x“11221100”, x”0000F102”, “Writing to Peripheral 1”, C\_SCOPE);  avalon\_mm\_write(AVALON\_MM\_VVCT, 1, C\_ADDR\_DMA, x”F102”, ”1111”, “Writing to DMA”, C\_SCOPE); |
| **avalon\_mm\_read()** | **avalon\_mm\_read(VVCT, vvc\_instance\_idx, addr, [TO\_SB,] msg, [scope])**  The avalon\_mm\_read() VVC procedure adds a read command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the read command is scheduled to run, the executor calls the Avalon-MM BFM avalon\_mm\_read() procedure, described in the Avalon-MM BFM QuickRef.  The value read from DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the read data will be stored in the VVC for a potential future fetch (see example below).  If the option TO\_SB is applied the read data will be sent to the AVALON\_MM\_VVC dedicated scoreboard where it will be checked against the expected value, which is provided by the testbench.  *Using read pipeline:*  If vvc\_config.use\_read\_pipeline has been set to true, the VVC will perform the read transaction using the BFM procedures avalon\_mm\_read\_request and avalon\_mm\_read\_response. First, the VVC executor will check if the number of pending commands in the pipeline will exceed the number of pipeline stages. If this is the case, the VVC executor will stall the read transaction until a command in the pipeline has been executed. The command executor will then let the BFM start the read request. After the read request has completed, the command will be added to the command response queue, which will run the BFM procedure avalon\_mm\_read\_response.    Example with fetch\_result() call (Result is placed in v\_data)  variable v\_cmd\_idx : natural;  variable v\_data : bitvis\_vip\_avalon\_mm.vvc\_cmd\_pkg.t\_vvc\_result;  (…)  avalon\_mm\_read(AVALON\_MM\_VVCT, 1, x“112252AA”, “Read from Peripheral 1”, C\_SCOPE);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(AVALON\_MM\_VVCT, 1); -- Store the command index (integer) for the last read  await\_completion(AVALON\_MM\_VVCT,1, v\_cmd\_idx, 100 ns, "Wait for read to finish");  fetch\_result(AVALON\_MM\_VVCT,1, v\_cmd\_idx, **v\_data**, "Fetching result from read operation"); |
| **avalon\_mm\_check()** | **avalon\_mm\_check(VVCT, vvc\_instance\_idx, addr, data, msg, [alert\_level, [scope]])**  The avalon\_mm\_check() VVC procedure adds a check command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the check command is scheduled to run, the executor calls the Avalon-MM BFM avalon\_mm\_check() procedure, described in the Avalon-MM BFM QuickRef. The avalon\_mm\_check() procedure will perform a read operation, then check if the read data is equal to the ‘data’ parameter. If the read data is not equal to the expected ‘data’ parameter, an alert with severity ‘alert\_level’ will be issued. The read data will not be stored by this procedure.  *Using read pipeline:*  If vvc\_config.use\_read\_pipeline has been set to true, the VVC will perform the check transaction using the BFM procedures avalon\_mm\_read\_request and avalon\_mm\_check\_response, similar to the procedure described in avalon\_mm\_read.  Example:  avalon\_mm\_check(AVALON\_MM\_VVCT, 1, x”11A49800”, x”0000393B”, “Check data from Peripheral 1”, ERROR, C\_SCOPE); |
| **avalon\_mm\_reset()** | **avalon\_mm\_reset(VVCT, vvc\_instance\_idx, num\_rst\_cycles, msg, [scope])**  The avalon\_mm\_reset() VVC procedure adds a reset command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the reset command is scheduled to run, the executor calls the Avalon-MM BFM avalon\_mm\_reset() procedure, described in the Avalon-MM BFM QuickRef.  Example:  avalon\_mm\_reset(AVALON\_MM\_VVCT, 1, 5, "Resetting Avalon MM Interface", C\_SCOPE); |
| **avalon\_mm\_lock()** | **avalon\_mm\_lock(VVCT, vvc\_instance\_idx, msg, [scope])**  The avalon\_mm\_lock() VVC procedure adds a lock command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the lock command is scheduled to run, the executor calls the Avalon-MM BFM avalon\_mm\_lock() procedure, described in the Avalon-MM BFM QuickRef.  Example:  avalon\_mm\_lock(AVALON\_MM\_VVCT, 1, "Locking Avalon MM Interface", C\_SCOPE); |
| **avalon\_mm\_unlock()** | **avalon\_mm\_unlock(VVCT, vvc\_instance\_idx, msg, [scope])**  The avalon\_mm\_unlock() VVC procedure adds an unlock command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the lock command is scheduled to run, the executor calls the Avalon-MM BFM avalon\_mm\_unlock() procedure, described in the Avalon-MM BFM QuickRef.  Example:  avalon\_mm\_unlock(AVALON\_MM\_VVCT, 1, "Locking Avalon MM Interface", C\_SCOPE); |

# VVC Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT** | **Description** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AVALON\_MM\_INTER\_BFM\_DELAY\_DEFAULT | Delay between any requested BFM accesses towards the DUT. - TIME\_START2START: Time from a BFM start to the next BFM start   (A TB\_WARNING will be issued if access takes   longer than TIME\_START2START).  - TIME\_FINISH2START: Time from a BFM end to the next BFM start. Any insert\_delay() command will add to the above minimum delays, giving for instance the ability to skew the BFM starting time. |
| cmd\_queue\_count\_max | natural | C\_MAX\_COMMAND\_QUEUE | Maximum pending number in command queue before queue is full. Adding additional commands will result in an ERROR. |
| cmd\_queue\_count\_threshold | natural | C\_CMD\_QUEUE\_COUNT\_THRESHOLD | An alert with severity “cmd\_queue\_count\_threshold\_severity” will be issued if command queue exceeds this count. Used for early warning if command queue is almost full. Will be ignored if set to 0. |
| cmd\_queue\_count\_threshold\_severity | t\_alert\_level | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | Severity of alert to be initiated if exceeding cmd\_queue\_count\_threshold |
| result\_queue\_count\_max | natural | C\_RESULT\_QUEUE\_COUNT\_MAX | Maximum number of unfetched results before result\_queue is full. |
| result\_queue\_count\_threshold | natural | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD | An alert with severity 'result\_queue\_count\_threshold\_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0. |
| result\_queue\_count\_threshold\_severity | t\_alert\_level | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | Severity of alert to be initiated if exceeding result\_queue\_count\_threshold |
| num\_pipeline\_stages | natural | 5 | Max read\_requests in pipeline |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | VVC dedicated message ID panel. See section 16 of uvvm\_vvc\_framework/doc/UVVM\_VVC\_Framework\_Essential\_Mechanisms.pdf for how to use verbosity control. |

The configuration record can be accessed from the Central Testbench Sequencer through the shared variable array, e.g.:

shared\_avalon\_mm\_vvc\_config(1).inter\_bfm\_delay.delay\_in\_time := 50 ns;

shared\_avalon\_mm\_vvc\_config(1).bfm\_config.use\_waitrequest := true;

# VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared\_avalon\_mm\_vvc\_status record from the test sequencer. The record contents can be seen below:

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **Description** |
| current\_cmd\_idx | natural | Command index currently running |
| previous\_cmd\_idx | natural | Previous command index to run |
| pending\_cmd\_cnt | natural | Pending number of commands in the command queue |

# Activity watchdog

The VVCs support a centralized VVC activity register which the activity watchdog uses to monitor the VVC activities. The VVCs will register their presence to the VVC activity register at start-up, and report when ACTIVE and INACTIVE, using dedicated VVC activity register methods, and trigger the global\_trigger\_vvc\_activity\_register signal during simulations. The activity watchdog is continuously monitoring the VVC activity register for VVC inactivity and raises an alert if no VVC activity is registered within the specified timeout period.

Include activity\_watchdog(num\_exp\_vvc, timeout, [alert\_level, [msg]]) in the testbench to start using the activity watchdog.   
Note that setting the exact number of expected VVCs in the VVC activity register can be omitted by setting num\_exp\_vvc = 0.

More information can be found in UVVM Essential Mechanisms PDF in the UVVM VVC Framework doc folder.

# Transaction Info

This VVC supports transaction info, a UVVM concept for distributing transaction information in a controlled manner within the complete testbench environment. The transaction info may be used in many different ways, but the main purpose is to share information directly from the VVC to a DUT model.

Table 5.1 Avalon MM transaction info record fields. Transaction type: t\_base\_transaction (BT) **-** accessiblevia **shared\_avalon\_mm\_vvc\_transaction\_info.bt**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Info field** | **Type** | **Default** | **Description** |
| operation | t\_operation | NO\_OPERATION | Current VVC operation, e.g. INSERT\_DELAY, POLL\_UNTIL, READ, WRITE. |
| addr | unsigned(63 downto 0) | 0x0 | Address of the AVALON MM read or write transaction. |
| data | slv(1023 downto 0) | 0x0 | Data for AVALON MM read or write transaction. |
| byte\_enable | slv(127 downto 0) | 0x0 | Used to indicate which bytes of data to use. When all bits are set to ‘1’, all bytes are enabled. |
| vvc\_meta | t\_vvc\_meta | C\_VVC\_META\_DEFAULT | VVC meta data of the executing VVC command. |
| **→** msg | string | “ “ | Message of executing VVC command. |
| **→** cmd\_idx | integer | -1 | Command index of executing VVC command. |
| transaction\_status | t\_transaction\_status | C\_TRANSACTION\_STATUS\_DEFAULT | Set to INACTIVE, IN\_PROGRESS, FAILED or SUCCEEDED during a transaction. |

Table 5.2 Avalon MM transaction info record fields. Transaction type: t\_sub\_transaction (ST) **-** accessiblevia **shared\_avalon\_mm\_vvc\_transaction\_info.sst**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Info field** | **Type** | **Default** | **Description** |
| operation | t\_operation | NO\_OPERATION | Current VVC operation, e.g. INSERT\_DELAY, POLL\_UNTIL, READ, WRITE. |
| addr | unsigned(63 downto 0) | 0x0 | Address of the AVALON MM read or write transaction. |
| data | slv(1023 downto 0) | 0x0 | Data for AVALON MM read or write transaction. |
| vvc\_meta | t\_vvc\_meta | C\_VVC\_META\_DEFAULT | VVC meta data of the executing VVC command. |
| **→** msg | string | “ “ | Message of executing VVC command. |
| **→** cmd\_idx | integer | -1 | Command index of executing VVC command. |
| transaction\_status | t\_transaction\_status | C\_TRANSACTION\_STATUS\_DEFAULT | Set to INACTIVE, IN\_PROGRESS, FAILED or SUCCEEDED during a transaction. |

See UVVM VVC Framework Essential Mechanisms PDF, section 6, for additional information about transaction types and transaction info usage.

# Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the TO\_SB parameter in supported method calls, i.e. avalon\_mm\_read(). Note that the data is only stored in the scoreboard and not accessible with the fetch\_result() method when the TO\_SB parameter is applied.

The Avalon MM VVC scoreboard is per default a 1024 bits wide standard logic vector. When sending data to the scoreboard, where the data width is smaller than the default scoreboard width, we recommend zero-padding the data with the pad\_avalon\_mm\_sb() function. E.g. AVALON\_MM\_VVC\_SB.add\_exepected(<Avalon MM instance number>, pad\_avalon\_mm\_sb(<v\_exp\_data>));

See the Generic Scoreboard Quick Reference PDF in the Bitvis VIP Scoreboard document folder for a complete list of available commands and additional information. The Avalon MM VVC scoreboard is accessible from the testbench as a shared variable AVALON\_MM\_VVC\_SB, located in the vvc\_methods\_pkg.vhd. All of the listed Generic Scoreboard commands are available for the Avalon MM VVC scoreboard using this shared variable.

# VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t\_avalon\_mm\_if* in order to improve readability of the code. Since the Avalon-MM interface busses can be of arbitrary size, the interface std\_logic\_vectors have been left unconstrained. These unconstrained SLVs needs to be constrained when the interface signals are instantiated. For this interface, this could look like:

signal avalon\_mm\_if\_1 : t\_avalon\_mm\_if( address(C\_ADDR\_WIDTH-1 downto 0),

byte\_enable((C\_DATA\_WIDTH/8)-1 downto 0),

writedata(C\_DATA\_WIDTH-1 downto 0),

readdata(C\_DATA\_WIDTH-1 downto 0) );

# Additional Documentation

Additional documentation about UVVM and its features can be found under “uvvm\_vvc\_framework/doc/”.

For additional documentation on the Avalon-MM standard, please see the Avalon specification “Avalon Interface Specifications, MNL-AVABUSREF”, available from Altera.

# Compilation

Avalon-MM VVC must be compiled with VHDL 2008.   
It is dependent on the following libraries

* ***UVVM Utility Library (UVVM-Util), version 2.15.0 and up***
* ***UVVM VVC Framework, version 2.10.0 and up***
* ***Avalon-MM BFM***
* ***Bitvis VIP Scoreboard***

Before compiling the Avalon-MM VVC, assure that uvvm\_vvc\_framework, uvvm\_util and bitvis\_vip\_scoreboard have been compiled.

See the UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc for information about compile scripts.

**Compile order for the Avalon-MM VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_avalon\_mm | avalon\_mm\_bfm\_pkg.vhd | Avalon-MM BFM |
| bitvis\_vip\_avalon\_mm | transaction\_pkg.vhd | Avalon-MM transaction package with DTT types, constants etc. |
| bitvis\_vip\_avalon\_mm | vvc\_cmd\_pkg.vhd | Avalon-MM VVC command types and operations |
| bitvis\_vip\_avalon\_mm | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the Avalon-MM VVC library. |
| bitvis\_vip\_avalon\_mm | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | UVVM VVC framework common methods compiled into the Avalon-MM VVC library |
| bitvis\_vip\_avalon\_mm | vvc\_methods\_pkg.vhd | Avalon-MM VVC methods |
| bitvis\_vip\_avalon\_mm | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_avalon\_mm | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the Avalon-MM VVC library |
| bitvis\_vip\_avalon\_mm | avalon\_mm\_vvc.vhd | Avalon-MM VVC |

# Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see ***UVVM-Util*** Quick reference.

IMPORTANT

This is a simplified Verification IP (VIP) for Avalon-MM. The given VIP complies with the basic Avalon-MM protocol and thus allows a normal access towards an Avalon-MM interface. This VIP is not an Avalon-MM protocol checker. For a more advanced VIP please contact Bitvis AS at [support@bitvis.no](mailto:support@bitvis.no)

Disclaimer: This IP and any part thereof are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.  
In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with this IP.

**INTELLECTUAL**

**PROPERTY**