**Avalon-Stream 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-Stream Master**

*avalon\_st\_vvc.vhd*

In order to use the Avalon-Stream VVC in master mode, it must be instantiated in the test harness by setting the generic constant *‘GC\_MASTER\_MODE’* to TRUE.

|  |
| --- |
| avalon\_st\_transmit (VVCT, vvc\_instance\_idx, [channel\_value], data\_array, msg, [scope]) |
| Example: avalon\_st\_transmit(AVALON\_ST\_VVCT, 0, v\_channel, v\_data\_array(0 to v\_numWords-1), “Send v\_numWords on v\_channel to DUT”, C\_SCOPE);  avalon\_st\_transmit(AVALON\_ST\_VVCT, 0, (x”01”, x”02”, x”03”, x”04”), “Send 4 bytes to DUT”);  *Note that this procedure can only be called when the AVALON\_ST VVC is instantiated in master mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to true.* |

**Avalon-Stream Slave**

In order to use the Avalon-Stream VVC in slave mode, it must be instantiated in the test harness by setting the generic constant *‘GC\_MASTER\_MODE’* to FALSE.

|  |
| --- |
| avalon\_st\_receive (VVCT, vvc\_instance\_idx, data\_array\_len, data\_word\_size, msg, [scope]) |
| Example: avalon\_st\_receive(AVALON\_ST\_VVCT, 1, v\_data\_array’length, v\_data\_array(0)’length, “Avalon ST Receive: Receive data will be stored in VVC. Retrieve later using fetch result() “);  avalon\_st\_receive(AVALON\_ST\_VVCT, 1, v\_data\_array’length, v\_data\_array(0)’length, “ Avalon ST Receive: Receive data will be sent to scoreboard “); |

|  |
| --- |
| avalon\_st\_expect (VVCT, vvc\_instance\_idx, [channel\_exp], data\_exp, msg, [alert\_level, [scope]]) |
| Example: avalon\_st\_expect(AVALON\_ST\_VVCT, 1, v\_channel, v\_data\_array(0 to v\_numWords-1), “Expect v\_numWords on v\_channel”, ERROR, C\_SCOPE);  avalon\_st\_expect(AVALON\_ST\_VVCT, 1, (x”01”, x”02”, x”03”, x”04”), “Expect 4 bytes”); |

 

|  |  |
| --- | --- |
| Avalon-Stream VVC Configuration record **´vvc\_config´ --** accessible via **shared\_avalon\_st\_vvc\_config** | **Common VVC procedures applicable for this VVC** |
| |  |  |  | | --- | --- | --- | | **Record element** | **Type** | **C\_AVALON\_ST\_VVC\_CONFIG\_DEFAULT** | | inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AVALON\_ST\_INTER\_BFM\_DELAY\_DEFAULT | | cmd\_queue\_count\_max | natural | C\_CMD\_QUEUE\_COUNT\_MAX | | cmd\_queue\_count\_threshold | natural | C\_CMD\_QUEUE\_COUNT\_THRESHOLD | | cmd\_queue\_count\_threshold\_severity | t\_alert\_level | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | | result\_queue\_count\_max | natural | C\_RESULT\_QUEUE\_COUNT\_MAX | | result\_queue\_count\_threshold | natural | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD | | result\_queue\_count\_threshold\_severity | t\_alert\_level | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | | bfm\_config | t\_avalon\_st\_bfm\_config | C\_AVALON\_ST\_BFM\_CONFIG\_DEFAULT | | msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | | unwanted\_activity\_severity | t\_alert\_level | C\_UNWANTED\_ACTIVITY\_SEVERITY | |  |  |  | | - See UVVM Methods QuickRef for details. |
| **await\_[any]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()** |
| Avalon-Stream VVC Status record signal **´vvc\_status´ --** accessible via **shared\_avalon\_st\_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\_ST\_VVCT | VVC target type compiled into each VVC in order to differentiate between VVCs. |
| vvc\_instance\_idx | integer | 0 | Instance number of the VVC |

VVC functional parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| channel\_value  channel\_exp | std\_logic\_vector | x”01” | Channel number for the data being transferred or expected.  The value is limited by max\_channel in the bfm\_config. |
| data\_array  data\_exp | t\_slv\_array | (x“D0D1”, x“D2D3”) | An array of SLVs containing the data to be sent/received.  data\_array(0) is sent/received first, while data\_array(data\_array’high) is sent/received last.  For clarity, data\_array is required to be ascending, for example defined by the test sequencer as follows:  variable v\_data\_array : t\_slv\_array(0 to C\_MAX\_WORDS-1)(C\_MAX\_WORD\_LENGTH-1 downto 0);  For simplicity, the word\_length can only be the size of the configured symbol (usually with packet-based transfers) or the size of the data bus (usually with data-based transfers).  variable v\_data\_array : t\_slv\_array(0 to C\_MAX\_WORDS-1)(C\_SYMBOL\_WIDTH-1 downto 0);  variable v\_data\_array : t\_slv\_array(0 to C\_MAX\_WORDS-1)(C\_DATA\_BUS\_LENGTH-1 downto 0); |
| data\_array\_len | natural | 20 | Length of the data\_array expected to be received (number of words). |
| data\_word\_size | natural | 8 | Size of the data words in the data\_array expected to be received. |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the procedure. |
| msg | string | “Send data” | A custom message to be appended in the log/alert |
| scope | string | “AVALON\_ST\_VVC” | A string describing the scope from which the log/alert originates.  In a simple single sequencer typically "AVALON\_ST\_BFM". In a verification component typically "AVALON\_ST\_VVC ". |

VVC entity signals

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

VVC entity generic constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| GC\_VVC\_IS\_MASTER | boolean | - | Set to true when this VVC instance is an Avalon-Stream master (data is output from BFM).  Set to false when this VVC is an Avalon-Stream slave (data is input to BFM.) |
| GC\_CHANNEL\_WIDTH | integer | 1 | Width of the Avalon-Stream channel signal.  *Note 1*: if CHANNEL is wider than 8, increase the value of the constant C\_AVALON\_ST\_CHANNEL\_MAX\_LENGTH in the local\_adaptations\_pkg.  *Note 2*: If the CHANNEL signal is not used, refer to description in Section 5. |
| GC\_DATA\_WIDTH | integer | - | Width of the Avalon-Stream data bus.  *Note*: if DATA is wider than 512, increase the value of the constant C\_AVALON\_ST\_WORD\_MAX\_LENGTH in the local\_adaptations\_pkg. |
| GC\_DATA\_ERROR\_WIDTH | integer | 1 | Width of the Avalon-Stream data error signal.  *Note*: If the DATA\_ERROR signal is not used, refer to description in Section 5. |
| GC\_EMPTY\_WIDTH | integer | 1 | Width of the Avalon-Stream empty signal.  *Note*: If the EMPTY signal is not used, refer to description in Section 5. |
| GC\_INSTANCE\_IDX | natural | - | Instance number to assign the VVC. |
| GC\_AVALON\_ST\_BFM\_CONFIG | t\_avalon\_st\_bfm\_config | C\_AVALON\_ST\_BFM\_  CONFIG\_DEFAULT | Configuration for the Avalon-Stream BFM, see Avalon-Stream 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 GC\_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

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **avalon\_st\_transmit()** | **avalon\_st\_transmit (VVCT, vvc\_instance\_idx, channel\_value, data\_array, msg, [scope])**  The avalon\_st\_transmit() VVC procedure adds a transmit command to the Avalon-Stream VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the Avalon-Stream BFM avalon\_st\_transmit() procedure, described in the Avalon-Stream BFM QuickRef.  The avalon\_transmit() procedure can only be called when the AVALON VVC is instantiated in master mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to true. |
| **avalon\_st\_receive()** | **avalon\_st\_receive (VVCT, vvc\_instance\_idx, data\_array\_len, data\_word\_size, msg, [scope])**  The avalon\_st\_receive() VVC procedure adds a receive command to the Avalon-Stream VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the Avalon-Stream BFM avalon\_st\_receive() procedure, described in the Avalon-Stream BFM QuickRef.  The avalon\_receive() procedure can only be called when the AVALON VVC is instantiated in slave mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to false.  The value received from the DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the received data and metadata will be stored in the VVC for a potential future fetch (see example with *fetch\_result* below).  **Example with fetch\_result() call**: Result is placed in **v\_result**  variable v\_cmd\_idx : natural; -- Command index for the last receive  variable v\_result : work.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from receive (data and metadata)  (…)  avalon\_st\_receive(AVALON\_ST\_VVCT, 1, v\_data\_array’length, v\_data\_array(0)’length, “Receive data in VVC”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(AVALON\_ST\_VVCT, 1);  await\_completion(AVALON\_ST\_VVCT, 1, 1 ms, “Wait for receive to finish”);  fetch\_result(AVALON\_ST\_VVCT, 1, v\_cmd\_idx, **v\_result**, “Fetching result from receive operation”); |
| **avalon\_st\_expect()** | **avalon\_st\_expect (VVCT, vvc\_instance\_idx, channel\_exp, data\_exp, msg, [alert\_level, [scope]])**  The avalon\_st\_expect() VVC procedure adds an expect command to the Avalon-Stream VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the Avalon-Stream BFM avalon\_st\_expect() procedure, described in the Avalon-Stream BFM QuickRef.  The avalon\_expect() procedure can only be called when the AVALON VVC is instantiated in slave mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to false. |
|  |  |

# VVC Instantiation

In order to select between the master and slave modes, the VVC must be instantiated using the correct value of the generic constant GC\_VVC\_IS\_MASTER in the testbench or test-harness. Example instantiations of the VVC in both operation supplied for ease of reference.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mode** | **Instantiation** |  | **Mode** | **Instantiation** |
| **Master** | i\_avalon\_st\_vvc\_master : **entity** work.avalon\_st\_vvc  **generic map**(  GC\_VVC\_IS\_MASTER => true,  GC\_CHANNEL\_WIDTH => GC\_CHANNEL\_WIDTH,  GC\_DATA\_WIDTH => GC\_DATA\_WIDTH,  GC\_DATA\_ERROR\_WIDTH => GC\_ERROR\_WIDTH,  GC\_EMPTY\_WIDTH => GC\_EMPTY\_WIDTH,  GC\_INSTANCE\_IDX => 0)  **port map**(  clk => clk,  avalon\_st\_vvc\_if => avalon\_st\_master\_if); |  | **Slave** | i\_avalon\_st\_vvc\_slave : **entity** work.avalon\_st\_vvc  **generic map**(  GC\_VVC\_IS\_MASTER => false,  GC\_CHANNEL\_WIDTH => GC\_CHANNEL\_WIDTH,  GC\_DATA\_WIDTH => GC\_DATA\_WIDTH,  GC\_DATA\_ERROR\_WIDTH => GC\_ERROR\_WIDTH,  GC\_EMPTY\_WIDTH => GC\_EMPTY\_WIDTH,  GC\_INSTANCE\_IDX => 1)  **port map**(  clk => clk,  avalon\_st\_vvc\_if => avalon\_st\_slave\_if); |

# VVC Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AVALON\_ST\_VVC\_CONFIG\_DEFAULT** | **Description** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AVALON\_ST\_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\_CMD\_QUEUE\_COUNT\_MAX | 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. |
| bfm\_config | t\_avalon\_st\_bfm\_config | C\_AVALON\_ST\_BFM\_CONFIG\_DEFAULT | Configuration for Avalon-Stream BFM. See quick reference for Avalon-Stream BFM. |
| 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. |
| unwanted\_activity\_severity | t\_alert\_level | C\_UNWANTED\_ACTIVITY\_SEVERITY | Severity of alert to be initiated if unwanted activity on the DUT outputs is detected. Unwanted activity detection is enabled (ERROR) by default. |

***Note: cmd/result queue parameters in the VVC Configuration are unused and will be removed in v3.0, use instead the entity generic constants.***

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

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

shared\_avalon\_st\_vvc\_config(1).bfm\_config.clock\_period := 10 ns;

# VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared\_avalon\_st\_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 6.1 Avalon Stream transaction info record fields. Transaction type: base transaction (BT) **-** accessiblevia **shared\_avalon\_st\_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. |
| channel\_value | slv(7 downto 0) | 0x0 | Channel number for the data being transferred or expected.  The value is limited by max\_channel in the bfm\_config.  The width of channel\_value can be configured through the local\_adaptations\_pkg by changing the value of C\_AVALON\_ST\_CHANNEL\_MAX\_LENGTH. Default value is 8. |
| data\_array | t\_slv\_array(0 to 1024)(512 downto 0) | (others => (others => ‘0’)) | An array of SLVs containing the data to be sent/received.  data\_array(0) is sent/received first, while data\_array(data\_array’high) is sent/received last.  The length of the data words, as well as the maximum amount of data words in data\_array, are configurable through the constants AVALON\_ST\_WORD\_MAX\_LENGTH and AVALON\_ST\_DATA\_MAX\_WORDS found in local\_adaptations\_pkg. |
| 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.

# VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t\_avalon\_st\_if* in order to improve readability of the code. Since the Avalon-Stream interface buses 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\_st\_if : t\_avalon\_st\_if (

channel(C\_CHANNEL\_WIDTH-1 downto 0),

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

data\_error(C\_ERROR\_WIDTH-1 downto 0),

empty(log2(C\_DATA\_WIDTH/C\_SYMBOL\_WIDTH)-1 downto 0));

The widths of *channel, data\_error* and *empty* are declared even when not used or connected to DUT.

Set the widths of unused signals to 1, for example C\_CHANNEL\_WIDTH = 1.

# Unwanted Activity Detection

This VVC supports detection of unwanted activity from the DUT. This mechanism will give an alert if the DUT generates any unexpected bus activity. It assures that no data is output from the DUT when it is not expected, i.e. Avalon-Stream receive/expect VVC methods are not called. Once the VVC is inactive, it starts to monitor continuously on the DUT outputs. When unwanted activity is detected, the VVC issues an alert of severity.

The unwanted activity detection can be configured from the central testbench sequencer, where the severity of alert can be changed to a different value.

To disable this feature in the testbench, e.g.:

shared\_avalon\_st\_vvc\_config(C\_VVC\_INDEX).unwanted\_activity\_severity := NO\_ALERT;

Note that the ready signal is not monitored in this VVC. The ready signal is allowed to be set independently of the valid signal, and there is no method to differentiate between the unwanted activity and intended activity. See the Avalon-Stream protocol specification for more information.

The unwanted activity detection is ignored when the valid signal goes low within one clock period after the VVC becomes inactive. This is to handle the situation when the read command exits before the next rising edge, causing signal transitions during the first clock cycle after the VVC is inactive.

For Avalon-ST VVC, the unwanted activity detection is enabled (unwanted\_activity\_severity := ERROR) by default.

# Additional Documentation

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

For additional documentation on the Avalon-Stream standard, refer to “Avalon® Interface Specifications, Chapter: Avalon Streaming Interfaces”, document number MNL-AVABUSREF, available from Intel.

# Compilation

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

* ***UVVM Utility Library (UVVM-Util), version 2.19.5 and up***
* ***UVVM VVC Framework, version 2.12.7 and up***
* ***Avalon-Stream BFM***

Before compiling the Avalon-Stream VVC, assure that uvvm\_vvc\_framework and uvvm\_util have been compiled.

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

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

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_avalon\_st | avalon\_st\_bfm\_pkg.vhd | Avalon-Stream BFM |
| bitvis\_vip\_avalon\_st | local\_adaptations\_pkg.vhd | Avalon-Stream adaptations package for local modifications. |
| bitvis\_vip\_avalon\_st | transaction\_pkg.vhd | Avalon-Stream transaction package with DTT types, constants etc. |
| bitvis\_vip\_avalon\_st | vvc\_cmd\_pkg.vhd | Avalon-Stream VVC command types and operations |
| bitvis\_vip\_avalon\_st | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the Avalon-Stream VVC library. |
| bitvis\_vip\_avalon\_st | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | UVVM framework common methods compiled into the Avalon-Stream VVC library |
| bitvis\_vip\_avalon\_st | vvc\_methods\_pkg.vhd | Avalon-Stream VVC methods |
| bitvis\_vip\_avalon\_st | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_avalon\_st | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the Avalon-Stream VVC library |
| bitvis\_vip\_avalon\_st | avalon\_st\_vvc.vhd | Avalon-Stream 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-Stream. The given VIP complies with the basic Avalon-Stream protocol and thus allows a normal access towards an Avalon-Stream interface. This VIP is not Avalon-Stream protocol checker. For a more advanced VIP please contact UVVM at [info@uvvm.org](mailto:info@uvvm.org)

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**