**AXI4-Stream VVC** –Quick Reference

**VVC**

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

**AXI4-Stream Master**

*axistream\_vvc.vhd*

In order to use the AXI4-Stream VVC in master mode, it must be instantiated in the test harness by setting the generic constant *‘GC\_MASTER\_MODE’* to TRUE.

|  |
| --- |
| axistream\_transmit[\_bytes] (VVCT, vvc\_instance\_idx, data\_array, [user\_array, [strb\_array, id\_array, dest\_array]], msg, [scope]) |
| Example: axistream\_transmit(AXISTREAM\_VVCT, 0, v\_data\_array(0 to v\_numBytes-1), v\_user\_array(0 to v\_numWords-1), “ Send a ‘v\_numBytes’ byte packet to DUT”);  axistream\_transmit(AXISTREAM\_VVCT, 0, v\_data\_array(0 to v\_numBytes-1)(31 downto 0), v\_user\_array(0 to v\_numWords-1), “Send a ‘4 x v\_numBytes’ byte packet to DUT”);  Note! Use axistream\_transmit\_bytes ( ) when using t\_byte\_array. |

**AXI4-Stream Slave**

In order to use the AXI4-Stream VVC in slave mode, it must be instantiated in the test harness by setting the generic constant *‘GC\_MASTER\_MODE’* to FALSE.

|  |
| --- |
| axistream\_receive[\_bytes] (VVCT, vvc\_instance\_idx, msg, [scope]) |
| Example: axistream\_receive (AXISTREAM\_VVCT, 1, “Receive packet, and store it in the VVC. To be fetched later using fetch\_result() “);  axistream\_receive (AXISTREAM\_VVCT, 1, “Receive packet, and send it to scoreboard for checking “);  Note! Use axistream\_receive\_bytes ( ) when using t\_byte\_array. |

|  |
| --- |
| axistream\_expect[\_bytes] (VVCT, vvc\_instance\_idx, exp\_data\_array, [exp\_user\_array, [exp\_strb\_array, exp\_id\_array, exp\_dest\_array]],   msg, [alert\_level, [scope]]) |
| Example: axistream\_expect(AXISTREAM\_VVCT, 0, v\_data\_array(0 to v\_numBytes-1), v\_user\_array(0 to v\_numWords-1), “Expect a packet, checking the tuser bits”);  axistream\_expect(AXISTREAM\_VVCT, 0, v\_data\_array(0 to v\_numBytes-1)(16 downto 0), v\_user\_array(0 to v\_numWords-1), “Expecting a packet, checking the tuser bits”);  Note! Use axistream\_expect\_bytes ( ) when using t\_byte\_array |



|  |  |
| --- | --- |
| AXI4-Stream VVC Configuration record **´vvc\_config´ --** accessible via **shared\_axistream\_vvc\_config** | **Common VVC procedures applicable for this VVC** |
| |  |  |  | | --- | --- | --- | | **Record element** | **Type** | **C\_AXISTREAM\_VVC\_CONFIG\_DEFAULT** | | inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AXISTREAM\_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 | | bfm\_config | t\_axistream\_bfm\_config | C\_AXISTREAM\_BFM\_CONFIG\_DEFAULT | | msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | |  |  |  | | - 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()** |
| AXI4-Stream VVC Status record signal **´vvc\_status´ --** accessible via **shared\_axistream\_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 | AXISTREAM\_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** |
| data\_array | t\_byte\_array, t\_slv\_array or  std\_logic\_vector | x”D0” & x“D1” (x”D0D1”, x”D2D3”) x”D0D1” | A byte array, SLV array or a single SLV containing the packet data to be sent or the data received.  Note the name change when data\_array is t\_byte\_array. SLV and t\_slv\_array data has to be a multiple of byte(s), e.g. x”AA”, x”BEEF”.  t\_byte\_array is defined in axistream\_bfm\_pkg. Refer to the AXI4-Stream BFM documentation |
| user\_array | t\_user\_array | x“1” & x“2” | Sideband data to send or has been received via the tuser signal.  t\_user\_array is defined in axistream\_bfm\_pkg. Refer to the AXI4-Stream BFM documentation |
| strb\_array | t\_strb\_array | x“1” & x“2” | Sideband data to send or has been received via the tstrb signal.  t\_strb\_array is defined in axistream\_bfm\_pkg. Refer to the AXI4-Stream BFM documentation |
| id\_array | t\_id\_array | x“1” & x“2” | Sideband data to send or has been received via the tid signal.  t\_id\_array is defined in axistream\_bfm\_pkg. Refer to the AXI4-Stream BFM documentation |
| dest\_array | t\_dest\_array | x“1” & x“2” | Sideband data to send or has been received via the tdest signal.  t\_dest\_array is defined in axistream\_bfm\_pkg. Refer to the AXI4-Stream BFM documentation |
| msg | string | “Send data” | 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 | “AXISTREAM VVC” | A string describing the scope from which the log/alert originates. In a simple single sequencer typically  "AXISTREAM BFM". In a verification component typically "AXISTREAM VVC ". |

VVC entity signals

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| clk | std\_logic | VVC Clock signal |
| axistream\_vvc\_master\_if | t\_axistream\_if | See AXI4-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 AXI4 Stream master (data is output from BFM).  Set to false when this VVC is an AXI4 Stream slave (data is input to BFM.) |
| GC\_DATA\_WIDTH | integer | - | Width of the AXI4-Stream data bus |
| GC\_USER\_WIDTH | integer | - | Width of the AXI4-Stream TUSER signal.­­  *Note 1*: if TUSER is wider than 8, increase the value of the constant C\_MAX\_TUSER\_BITS in axistream\_bfm\_pkg.  *Note 2*: If the TUSER signal is not used, refer to description in Section 7 |
| GC\_ID\_WIDTH | integer | - | Width of the AXI4-Stream TID signal.­­  *Note 1*: if TID is wider than 8, increase the value of the constant C\_MAX\_TID\_BITS in axistream\_bfm\_pkg.  *Note 2*: If the TID signal is not used, refer to description in Section 7 |
| GC\_DEST\_WIDTH | integer | - | Width of the AXI4-Stream TDEST signal.­­  *Note 1*: if TDEST is wider than 4, increase the value of the constant C\_MAX\_TDEST\_BITS in axistream\_bfm\_pkg.  *Note 2*: If the TDEST signal is not used, refer to description in Section 7 |
| GC\_INSTANCE\_IDX | natural | - | Instance number to assign the VVC |
| GC\_AXISTREAM\_CONFIG | t\_axistream\_bfm\_config | C\_AXISTREAM\_BFM\_  CONFIG\_DEFAULT | Configuration for the AXI4-Stream BFM, see AXI4-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 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 the procedure name change when using t\_byte\_array.

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

# VVC procedure details

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **axistream\_transmit[\_bytes]()** | **axistream\_transmit[\_bytes] (VVCT, vvc\_instance\_idx, data\_array, [user\_array, [strb\_array, id\_array, dest\_array]], msg, [scope])**  The axistream\_transmit() VVC procedure adds a transmit command to the AXI4-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 AXI4-Stream BFM axistream\_transmit() procedure, described in the AXI4-Stream BFM QuickRef.  The axistream\_transmit() procedure can only be called when the AXISTREAM VVC is instantiated in master mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to true.  Examples:  axistream\_transmit(AXISTREAM\_VVCT, 0, v\_data\_array(0 to 1),  “Send a 2 byte packet to DUT, tuser=0 each word / clock cycle”, C\_SCOPE);  axistream\_transmit(AXISTREAM\_VVCT, 0, v\_data\_array(0 to 1)(16 downto 0),   “Send a 4 byte packet to DUT, tuser=0 each word / clock cycle”, C\_SCOPE);  axistream\_transmit(AXISTREAM\_VVCT, 0, v\_data\_array(0 to v\_numBytes-1), v\_user\_array(0 to v\_numWords-1),   “ Send a ‘v\_numBytes’ byte packet to DUT”, C\_SCOPE);  axistream\_transmit(AXISTREAM\_VVCT, 0, v\_data\_array(0 to v\_numBytes-1), v\_user\_array(0 to v\_numWords-1),   v\_strb\_array(0 to v\_numWords-1), v\_id\_array(0 to v\_numWords-1),   v\_id\_array(0 to v\_numWords-1), “Send..”, C\_SCOPE);  axistream\_transmit(AXISTREAM\_VVCT, 0, (x”D0”, x”D1”, x”D2”, x”D3”), (x”00”, x”0A”),   “Send a 4 byte packet with tuser=A at the 2nd (last) word”, C\_SCOPE); --(tdata’length = 16)  axistream\_transmit(AXISTREAM\_VVCT, 0, (x”D0”, x”D1”, x”D2”, x”D3”), (x”00”, x”00”, x”00”, x”0A”),   “Send a 4 byte packet with tuser=A at the 4th (last) word”, C\_SCOPE); --(tdata’length = 8) |
| **axistream\_expect[\_bytes]()** | **axistream\_expect[\_bytes] (VVCT, vvc\_instance\_idx, exp\_data\_array, [exp\_user\_array, [exp\_strb\_array, exp\_id\_array, exp\_dest\_array]],   msg, [alert\_level, [scope]])**  The axistream\_expect() VVC procedure adds an expect command to the AXI4-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 AXI4-Stream BFM axistream\_expect() procedure, described in the AXI4-Stream BFM QuickRef.  The axistream\_expect() procedure can only be called when the AXISTREAM VVC is instantiated in slave mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to false.  Examples:  axistream\_expect(AXISTREAM\_VVCT, 0, v\_exp\_data\_array(0 to 1),   “Expect a 2 byte packet, ignoring the tuser bits”, ERROR, C\_SCOPE);  axistream\_expect(AXISTREAM\_VVCT, 0, v\_exp\_data\_array(0 to 1)(16 downto 0),  “Expect a 4 byte packet, ignoring the tuser bits”, ERROR, C\_SCOPE);  axistream\_expect(AXISTREAM\_VVCT, 0, v\_exp\_data\_array(0 to v\_numBytes-1), v\_user\_array(0 to v\_numWords-1),   “Expect a packet, checking the tuser bits”, ERROR, C\_SCOPE);  axistream\_expect(AXISTREAM\_VVCT, 0, v\_exp\_data\_array(0 to v\_numBytes-1), v\_user\_array(0 to v\_numWords-1),   v\_strb\_array(0 to v\_numWords-1), v\_id\_array(0 to v\_numWords-1),v\_id\_array(0 to v\_numWords-1),   “Check all sigs”, ERROR, C\_SCOPE);  axistream\_expect(AXISTREAM\_VVCT, 0, (x”D0”, x”D1”, x”D2”, x”D3”), (x”00”, x”0A”), “Expect a 4 byte packet with   tuser=A at the 2nd (last) word”, ERROR, C\_SCOPE); --(tdata’length = 16)  axistream\_expect(AXISTREAM\_VVCT, 0, (x”D0”, x”D1”, x”D2”, x”D3”), (x”00”, x”00”, x”00”, x”0A”), “Expect a   4 byte packet with tuser=A at the 4th (last) word”, ERROR, C\_SCOPE); --(tdata’length = 8) |
| **axistream\_receive[\_bytes] ()** | **axistream\_receive[\_bytes] (VVCT, vvc\_instance\_idx, msg, [scope])**  The axistream\_receive() VVC procedure adds a receive command to the AXISTREAM VVC executor queue, which will run as soon as all preceding commands have completed. When the receive command is scheduled to run, the executor calls the AXISTREAM BFM axistream\_receive() procedure, described in the AXISTREAM BFM QuickRef. The axistream\_reveice() procedure can only be called when the AXISTREAM VVC is instantiated in slave mode, i.e. setting the generic constant ‘GC\_MASTER\_MODE’ to false.  The value receive from 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).  Note that the stored received data is t\_byte\_array.  Example:  axistream\_receive(AXISTREAM\_VVCT, 1, “Receive data to VVC”, C\_SCOPE);  **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)  (…)  axistream\_receive(AXISTREAM\_VVCT, 1, “Receive data to VVC”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(AXISTREAN\_VVCT, 1);  await\_completion(AXISTREAM\_VVCT,1, 1 ms, “Wait for receive to finish”);  fetch\_result(AXISTREAM\_VVCT,1, v\_cmd\_idx, **v\_result**, “Fetching result from receive operation”); |
|  |  |

# 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** | **Instatiation** |  | **Mode** | **Instatiation** |
| **Master** | i\_axistream\_vvc\_master: **entity** work.axistream\_vvc  **generic map**(  GC\_VVC\_IS\_MASTER => true,  GC\_DATA\_WIDTH => GC\_DATA\_WIDTH,  GC\_USER\_WIDTH => GC\_USER\_WIDTH,  GC\_ID\_WIDTH => GC\_ID\_WIDTH,  GC\_DEST\_WIDTH => GC\_DEST\_WIDTH,  GC\_INSTANCE\_IDX => 2)  **port map**(  clk => clk,  axistream\_vvc\_if => axistream\_if); |  | **Slave** | i\_axistream\_vvc\_slave : **entity** work.axistream\_vvc  **generic map**(  GC\_VVC\_IS\_MASTER => false,  GC\_DATA\_WIDTH => GC\_DATA\_WIDTH,  GC\_USER\_WIDTH => GC\_USER\_WIDTH,  GC\_ID\_WIDTH => GC\_ID\_WIDTH,  GC\_DEST\_WIDTH => GC\_DEST\_WIDTH,  GC\_INSTANCE\_IDX => 3)  **port map**(  clk => clk,  axistream\_vvc\_if => axistream\_if); |

# VVC Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AXISTREAM\_BFM\_CONFIG\_DEFAULT** | **Description** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AXISTREAM\_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\_axistream\_bfm\_config | C\_AXISTREAM\_BFM\_CONFIG\_DEFAULT | Configuration for AXI4-Stream BFM. See quick reference for AXI4-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. |

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

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

shared\_axistream\_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\_axistream\_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.

See UVVM VVC Framework Essential Mechanisms PDF, section 6, for additional information.

Table 6.1 AXI4-Stream base transaction (BT) record fields. Transaction type: t\_base\_transaction (BT) **-** accessiblevia **shared\_axistream\_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. |
| data\_array | t\_slv\_array(0 to 16\*1024) | (others => (others => '0')) | Packet data to be sent or received. |
| user\_array | t\_user\_array(0 to 16\*1024) | (others => (others => '0')) | Sideband data to send or which has been received via the tuser signal. |
| strb\_array | t\_strb\_array(0 to 16\*1024) | (others => (others => '0')) | Sideband data to send or which has been received via the tstrb signal. |
| id\_array | t\_id\_array(0 to 16\*1024) | (others => (others => '0')) | Sideband data to send or which has been received via the tid signal. |
| dest\_array | t\_dest\_array(0 to 16\*1024) | (others => (others => '0')) | Sideband data to send or which has been received via the tdest signal. |
| 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. |

Refer to the the VVC Functional Parameters table in page 3 for more details regarding the VVC specific Transaction Info record fields. 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\_axistream\_if* in order to improve readability of the code. Since the AXI4-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, they could look like:

signal axistream\_if : t\_axistream\_if(tdata(C\_DATA\_WIDTH -1 downto 0),

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

tuser(C\_USER\_WIDTH -1 downto 0),

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

tid(C\_ID\_WIDTH-1 downto 0),

tdest(C\_DEST\_WIDTH-1 downto 0)

);

The widths of *tuser, tstrb, tid* and *tdest* are declared even when not used or connected to DUT.

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

# Additional Documentation

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

For additional documentation on the AXI4-Stream standard, refer to “AMBA 4 AXI4-Stream Protocol Specification (ARM IHI 0051)”, available from ARM.

# Compilation

AXI4-Stream VVC must be compiled with VHDL 2008.   
It is dependent on the following libraries

* ***UVVM Utility Library (UVVM-Util), version 2.14.0 and up***
* ***UVVM VVC Framework, version 2.10.0 and up***
* ***AXI4-Stream BFM***

Before compiling the AXI4-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 AXI4-Stream VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_axistream | axistream\_bfm\_pkg.vhd | AXI4-Stream BFM |
| bitvis\_vip\_axistream | transaction\_pkg.vhd | AXI4-Stream transaction package with DTT types, constants etc. |
| bitvis\_vip\_axistream | vvc\_cmd\_pkg.vhd | AXI4-Stream VVC command types and operations |
| bitvis\_vip\_axistream | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the AXI4-Stream VVC library. |
| bitvis\_vip\_axistream | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | UVVM framework common methods compiled into the AXI4-Stream VVC library |
| bitvis\_vip\_axistream | vvc\_methods\_pkg.vhd | AXI4-Stream VVC methods |
| bitvis\_vip\_axistream | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_axistream | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the AXI4-Stream VVC library |
| bitvis\_vip\_axistream | axistream\_vvc.vhd | AXI4-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 AXI4-Stream. The given VIP complies with the basic AXI4-Stream protocol and thus allows a normal access towards an AXI4-Stream interface. This VIP is not AXI4-Stream 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**