**AXI4 VVC** –Quick Reference

**VVC**

For general information see UVVM VVC Framework Essential Mechanisms located in uvvm\_vvc\_framework/doc.

*axi\_vvc.vhd*

|  |
| --- |
| axi\_write (VVCT, vvc\_instance\_idx, awid, awaddr, awlen, awsize, awburst, awlock, awcache, awprot, awqos, awregion, awuser, wdata, wstrb, wuser, bresp\_exp, buser\_exp, msg, [scope]) |
| Example: axi\_write(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  awid => x”01”,  awaddr => x"00000004",  awlen => x"01",  awsize => 4,  awburst => INCR,  awlock => NORMAL,  awcache => "0000",  awprot => UNPRIVILEGED\_UNSECURE\_DATA,  awqos => "0000",  awregion => "0000",  awuser => x"01",  wdata => t\_slv\_array'(x"12345678", x"33333333"),  wstrb => t\_slv\_array'(x"F", x"F"),  wuser => t\_slv\_array'(x"01", x"01"),  buser\_exp => OKAY,  bresp\_exp => x"01",  msg => "Writing data to Peripheral 1"); |



|  |
| --- |
| axi\_read (VVCT, vvc\_instance\_idx, arid, araddr, arlen, arsize, airburst, arlock, arcache, arprot, arqos, arregion, aruser, data\_routing, msg, [scope]) |
| Example: axi\_read(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  arid => x"01",  araddr => x"00000004",  arlen => x"01",  arsize => 4,  arburst => INCR,  arlock => NORMAL,  arcache => "0000",  arprot => UNPRIVILEGED\_UNSECURE\_DATA,  arqos => "0000",  arregion => "0000",  aruser => x"01",  data\_routing => TO\_SB,  msg => "Read from Peripheral 1 and send result to scoreboard"); |

|  |
| --- |
| axi\_check (VVCT, vvc\_instance\_idx, arid, araddr, arlen, arsize, airburst, arlock, arcache, arprot, arqos, arregion, aruser, rdata\_exp, rresp\_exp, ruser\_exp, msg, [alert\_level, [scope]]) |
| Example: axi\_check(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  arid => x"01",  araddr => x"00000004",  arlen => x"01",  arsize => 4,  arburst => INCR,  arlock => NORMAL,  arcache => "0000",  arprot => UNPRIVILEGED\_UNSECURE\_DATA,  arqos => "0000",  arregion => "0000",  aruser => x"01",  rdata\_exp => t\_slv\_array'(x"12345678", x"33333333"),  rresp\_exp => t\_xresp\_array'(OKAY, OKAY),  ruser\_exp => t\_slv\_array'(x"00", x"00"),  msg => "Check data from Peripheral 1"); |

AXI4 VVC Configuration record **´vvc\_config´ --** accessible via **shared\_axi\_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\_AXI\_VVC\_CONFIG\_DEFAULT** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AXI\_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 |
| bfm\_config | t\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT |
|  |  |  |

AXI4 VVC Status record signal **´vvc\_status´ --** accessible via **shared\_axi\_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 | AXI\_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** |
| axid | std\_logic\_vector | x”01” | Identification tag for a read or write transaction |
| axaddr | unsigned | x”325A” | The address of the first transfer in a read or write transaction |
| axlen | unsigned(7 downto 0) | x”01” | The number of data transfers in a read or write transactions |
| axsize | Integer range 1 to 128 | 4 | The number of bytes in each data transfer in a read or write transaction (Must be a power of two) |
| axburst | t\_axburst | INCR | Burst type, indicates how address changes between each transfer in a read or write transaction |
| axlock | t\_axlock | NORMAL | Provides information about the atomic characteristics of a read or write transaction |
| axcache | std\_logic\_vector(3 downto 0) | “0000” | Indicates how a read or write transaction is required to progress through a system |
| axprot | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | Protection attributes of a read or write transaction. Privilege, security level and access type |
| axqos | std\_logic\_vector(3 downto 0) | “0000” | Quality of Service identifier for a read or write transaction |
| axregion | std\_logic\_vector(3 downto 0) | “0000” | Region indicator for a read or write transaction |
| axuser | std\_logic\_vector | x”01” | User-defined extension for the read or write address channel |
| wdata | t\_slv\_array | t\_slv\_array’(x”20D3”, x”1234”) | Array of data values to be written to the addressed registers |
| wstrb | t\_slv\_array | t\_slv\_array’(”1111”, ”1111”) | Array of write strobes, indicates which byte lanes hold valid data. (all ‘1’ means all bytes are updated) |
| wuser | t\_slv\_array | t\_slv\_array’(x”00”, x”01”) | Array of user-defined extension for the write data channel |
| bresp\_exp | t\_xresp | OKAY | Expected write response which indicates the status of a write transaction |
| buser\_exp | std\_logic\_vector | x”01” | Expected user-defined extension for the write response channel |
| rdata\_exp | t\_slv\_array | t\_slv\_array’(x”ABCD”, x”1234”) | Array of expected read data values. A mismatch results in an alert ‘alert\_level’ |
| rresp\_exp | t\_xresp\_array | t\_xresp\_array’(OKAY, OKAY) | Array of expected read responses which indicates the status of a read transfer. A mismatch results in an alert ‘alert\_level’ |
| ruser\_exp | t\_slv\_array | t\_slv\_array’(x”01”, x”01”) | Array of expected user-defined extensions for the read data channel. A mismatch results in an alert ‘alert\_level’ |
| data\_routing | t\_data\_routing | TO\_SB | Selects the destination of the read data. Scoreboard: TO\_SB or read buffer: TO\_BUFFER |
| 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 | “AXI VVC” | A string describing the scope from which the log/alert originates. In a simple single sequencer typically  "AXI BFM". In a verification component typically "AXI VVC ". |

VVC entity signals

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| clk | std\_logic | VVC Clock signal |
| axi\_vvc\_master\_if | t\_axi\_if | See AXI4 BFM documentation |

VVC entity generic constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| GC\_ADDR\_WIDTH | integer | 8 | Width of the AXI4 address bus (AWADDR, ARADDR) |
| GC\_DATA\_WIDTH | integer | 32 | Width of the AXI4 data bus (WDATA, RDATA). The write strobe (WSTRB) is derived from this (GC\_DATA\_WIDTH/8) |
| GC\_ID\_WIDTH | integer | 8 | Width of the AXI4 ID signals (AWID, BID, ARID, RID) |
| GC\_USER\_WIDTH | integer | 8 | Width of the AXI4 User signals (AWUSER, WUSER, BUSER, ARUSER, RUSER) |
| GC\_INSTANCE\_IDX | natural | 1 | Instance number to assign the VVC |
| GC\_AXI\_CONFIG | t\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT | Configuration for the AXI4 BFM, see AXI4 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** |
| **axi\_write()** | **axi\_write(VVCT, vvc\_instance\_idx, awid, awaddr, awlen, awsize, awburst, awlock, awcache, awprot, awqos, awregion, awuser, wdata, wstrb, wuser, bresp\_exp, buser\_exp, msg, [scope])**  The axi\_write() VVC procedure adds a write command to the AXI4 VVC executor queue, which will distribute this command to the various channel executors which in turn will run as soon as all preceding commands have completed. When the write command is scheduled to run, the executors call the AXI4 procedures in axi\_channel\_handler\_pkg.vhd.  axi\_write can be called with or without parameters that already have a default value.   |  |  |  | | --- | --- | --- | | **Parameter name** | **Type** | **Default value** | | VVCT | t\_vvc\_target\_record | None | | vvc\_instance\_idx | integer | None | | awid | std\_logic\_vector | 0 | | awaddr | unsigned | None | | awlen | unsigned(7 downto 0) | 0 | | awsize | Integer range 1 to 128 | 4 | | awburst | t\_axburst | INCR | | awlock | t\_axlock | NORMAL | | awcache | std\_logic\_vector(3 downto 0) | 0 | | awprot | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | | awqos | std\_logic\_vector(3 downto 0) | 0 | | awregion | std\_logic\_vector(3 downto 0) | 0 | | awuser | std\_logic\_vector | 0 | | wdata | t\_slv\_array | None | | wstrb | t\_slv\_array | (others => ‘1’) for all words | | wuser | t\_slv\_array | 0 for all words | | bresp\_exp | std\_logic\_vector(1 downto 0) | OKAY | | buser\_exp | std\_logic\_vector | 0 | | msg | string | None | | scope | string | “TB seq.(uvvm)” |     Examples:  axi\_write(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  awid => x"01",  awaddr => x"00000004",  awlen => x"01",  awsize => 4,  awburst => INCR,  awlock => NORMAL,  awcache => "0000",  awprot => UNPRIVILEGED\_UNSECURE\_DATA,  awqos => "0000",  awregion => "0000",  awuser => x"01",  wdata => t\_slv\_array'(x"12345678", x"33333333"),  wstrb => t\_slv\_array'(x"F", x"F"),  wuser => t\_slv\_array'(x"01", x"01"),  bresp\_exp => OKAY,  buser\_exp => x"00",  msg => "Writing data to Peripheral 1");  axi\_write(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  awaddr => x"00000004",  wdata => t\_slv\_array'(x"12345678", x"33333333"),  msg => "Writing data to Peripheral 1"); |
| **axi\_read()** | **axi\_read(VVCT, vvc\_instance\_idx, arid, araddr, arlen, arsize, airburst, arlock, arcache, arprot, arqos, arregion, aruser, data\_routing, msg, [scope])**  The axi\_read() VVC procedure adds a read command to the AXI4 VVC executor queue, which will distribute this command to the various channel executors which in turn will run as soon as all preceding commands have completed. When the read command is scheduled to run, the executors call the AXI4 procedures in axi\_channel\_handler\_pkg.vhd.  The value read from the DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller. If the data\_routing parameter is set to TO\_BUFFER, the read data will be stored in the VVC for a potential future fetch (see example with fetch\_result() below). If the data\_routing parameter is set to TO\_SB, the received data will be sent to the AXI VVC dedicated scoreboard where it will be checked against the expected value (provided by the testbench)   |  |  |  | | --- | --- | --- | | **Parameter name** | **Type** | **Default value** | | VVCT | t\_vvc\_target\_record | None | | vvc\_instance\_idx | integer | None | | arid | std\_logic\_vector | 0 | | araddr | unsigned | None | | arlen | unsigned(7 downto 0) | 0 | | arsize | Integer range 1 to 128 | 4 | | arburst | t\_axburst | INCR | | arlock | t\_axlock | NORMAL | | arcache | std\_logic\_vector(3 downto 0) | 0 | | arprot | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | | arqos | std\_logic\_vector(3 downto 0) | 0 | | arregion | std\_logic\_vector(3 downto 0) | 0 | | aruser | std\_logic\_vector | 0 | | data\_routing | t\_data\_routing | None | | msg | string | None | | scope | string | “TB seq.(uvvm)” |   Examples:  axi\_read(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  arid => x“01”,  araddr => x”00000004”,  arlen => x”01”,  arsize => 4,  arburst => INCR,  arlock => NORMAL,  arcache => “0000”,  arprot => UNPRIVILEGED\_UNSECURE\_DATA,  arqos => “0000”,  arregion => “0000”,  aruser => x“01”,  data\_routing => TO\_SB,  msg => “Read from Peripheral 1 and send result to scoreboard”);  axi\_read(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  araddr => x”00000004”,  data\_routing => TO\_BUFFER,  msg => “Read from Peripheral 1 and send result to read buffer”);  **Example with fetch\_result() call. Result is placed in v\_result**  variable v\_cmd\_idx : natural; -- Command index for the last read  variable v\_result : work.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from read  (…)  axi\_read(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  araddr => x”00000004”,  data\_routing => TO\_BUFFER,  msg => “Read from Peripheral 1 and send result to read buffer”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(AXI\_VVCT, 1);  await\_completion(AXI\_VVCT, 1, 100 ns, "Wait for read to finish");  fetch\_result(AXI\_VVCT,1, v\_cmd\_idx, **v\_result**, "Fetching result from read operation"); |
| **axi\_check()** | **axi\_check(VVCT, vvc\_instance\_idx, arid, araddr, arlen, arsize, airburst, arlock, arcache, arprot, arqos, arregion, aruser, rdata\_exp, rresp\_exp, ruser\_exp, msg, [alert\_level, [scope]])**  The axi\_check() VVC procedure adds a check command to the AXI4 VVC executor queue, which will distribute this command to the various channel executors which in turn will run as soon as all preceding commands have completed. When the check command is scheduled to run, the executors call the AXI4 procedures in axi\_channel\_handler\_pkg.vhd. The axi\_check() procedure will perform a read operation, then check if the read result is equal to the rdata\_exp, rresp\_exp and ruser\_exp parameters. If the result is not equal to the expected result, an alert with severity ‘alert\_level’ will be issued. The read data will not be stored by this procedure.   |  |  |  | | --- | --- | --- | | **Parameter name** | **Type** | **Default value** | | VVCT | t\_vvc\_target\_record | None | | vvc\_instance\_idx | integer | None | | arid | std\_logic\_vector | 0 | | araddr | unsigned | None | | arlen | unsigned(7 downto 0) | 0 | | arsize | Integer range 1 to 128 | 4 | | arburst | t\_axburst | INCR | | arlock | t\_axlock | NORMAL | | arcache | std\_logic\_vector(3 downto 0) | 0 | | arprot | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | | arqos | std\_logic\_vector(3 downto 0) | 0 | | arregion | std\_logic\_vector(3 downto 0) | 0 | | aruser | std\_logic\_vector | 0 | | rdata\_exp | t\_slv\_array | None | | rresp\_exp | t\_xresp\_array | OKAY for all words | | ruser\_exp | t\_slv\_array | 0 for all words | | msg | string | None | | alert\_level | t\_alert\_level | ERROR | | scope | string | “TB seq.(uvvm)” |   Examples:  axi\_check(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  arid => x"01",  araddr => x"00000004",  arlen => x"01",  arsize => 4,  arburst => INCR,  arlock => NORMAL,  arcache => "0000",  arprot => UNPRIVILEGED\_UNSECURE\_DATA,  arqos => "0000",  arregion => "0000",  aruser => x"01",  rdata\_exp => t\_slv\_array'(x"12345678", x"33333333"),  rresp\_exp => t\_xresp\_array'(OKAY, OKAY),  ruser\_exp => t\_slv\_array'(x"00", x"00"),  msg => "Check data from Peripheral 1");  axi\_check(  VVCT => AXI\_VVCT,  vvc\_instance\_idx => 1,  araddr => x"00000004",  rdata\_exp => t\_slv\_array'(x"12345678", x"33333333"),  msg => "Check data from Peripheral 1"); |

# VVC Configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AXI\_VVC\_CONFIG\_DEFAULT** | **Description** | |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AXI\_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  - TIME\_FINISH2START: Not supported by this VVC 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 |
| bfm\_config | t\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT | Configuration for AXI4 BFM. See quick reference for AXI4 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\_axi\_vvc\_config(1).inter\_bfm\_delay.delay\_in\_time := 50 ns;

shared\_axi\_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\_axi\_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 AXI4 transaction info record fields. Transaction type: t\_base\_transaction (BT) **-** accessiblevia **shared\_axi\_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(31 downto 0) | 0x0 | Address of the AXI read or write transaction. |
| data | slv(255 downto 0) | 0x0 | Data for AXI check or write transaction. |
| byte\_enable | slv(31 downto 0) | 0x0 | Byte enable for the AXI 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 data\_routing parameter to TO\_SB in supported method calls, i.e. axi\_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 result which is stored in the scoreboard is of the type t\_vvc\_result which is detailed below.

Table 5.2 t\_vvc\_result type

|  |  |
| --- | --- |
| **Record element** | **Type** |
| len | natural range 0 to 255 |
| rid | std\_logic\_vector(31 downto 0) |
| rdata | t\_slv\_array(0 to 255)(255 downto 0) |
| rresp | t\_xresp\_array(0 to 255) |
| ruser | t\_slv\_array(0 to 255)(127 downto 0) |

The AXI VVC scoreboard is per default the maximum width of rid, rdata, rresp and ruser. When sending expected result to the scoreboard, where the result width is smaller than the default scoreboard width, we recommend zero-padding the 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 AXI4 VVC scoreboard is accessible from the testbench as a shared variable AXI\_VVC\_SB, located in the vvc\_methods\_pkg.vhd. All of the listed Generic Scoreboard commands are available for the AXI4 VVC scoreboard using this shared variable.

# VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t\_axi\_if* to improve readability of the code. Since the AXI4 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 axi\_if : t\_axi\_if( write\_address\_channel( awid( C\_ID\_WIDTH -1 downto 0),

awaddr( C\_ADDR\_WIDTH -1 downto 0),

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

write\_data\_channel ( wdata( C\_DATA\_WIDTH -1 downto 0),

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

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

write\_response\_channel(bid( C\_ID\_WIDTH -1 downto 0),

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

read\_address\_channel ( arid( C\_ID\_WIDTH -1 downto 0),

araddr( C\_ADDR\_WIDTH -1 downto 0),

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

read\_data\_channel ( rid( C\_ID\_WIDTH -1 downto 0),

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

ruser( C\_USER\_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 AXI4 standard, please see the AXI4 specification “AMBA® AXI™ and ACE™ Protocol

Specification”, available from ARM.

# Compilation

AXI4 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 BFM***
* ***Bitvis VIP Scoreboard***

Before compiling the AXI4 VVC, assure that uvvm\_vvc\_framework, uvvm\_util and bitvis\_vip\_scoreboard have been compiled.

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

**Compile order for the AXI4 VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_axi | axi\_bfm\_pkg.vhd | AXI4 BFM |
| bitvis\_vip\_axi | transaction\_pkg | AXI4 transaction package with DTT types, constants etc. |
| bitvis\_vip\_axi | vvc\_cmd\_pkg.vhd | AXI4 VVC command types and operations |
| bitvis\_vip\_axi | axi\_read\_data\_queue\_pkg.vhd | Package for storing read data responses in a queue |
| bitvis\_vip\_axi | axi\_channel\_handler\_pkg.vhd | Package containing procedures for accessing AXI4 channels |
| bitvis\_vip\_axi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the AXI4 VVC library. |
| bitvis\_vip\_axi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | UVVM framework common methods compiled into the AXI4 VVC library |
| bitvis\_vip\_axi | axi\_sb\_pkg.vhd | AXI4 scoreboard package (instantiating the generic scoreboard) |
| bitvis\_vip\_axi | vvc\_methods\_pkg.vhd | AXI4 VVC methods |
| bitvis\_vip\_axi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_axi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the AXI4 VVC library |
| bitvis\_vip\_axi | axi\_vvc.vhd | AXI4 VVC |
| bitvis\_vip\_axi | vvc\_context.vhd | AXI4 VVC context |

# 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. The given VIP complies with the AXI4 protocol and thus allows a normal access towards an AXI4 interface. This VIP is not AXI4 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**