**AXI4-Lite VVC** –Quick Reference

**VVC**

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

|  |
| --- |
| axilite\_write (VVCT, vvc\_instance\_idx, addr, data, [byte\_enable], msg, [scope]) |
| Example: axilite\_write(AXILITE\_VVCT, 1, x"6000", x”F102”, “Writing data to Peripheral 1”); |

*axilite\_vvc.vhd*

|  |
| --- |
| axilite\_read (VVCT, vvc\_instance\_idx, addr, msg, [scope]) |
| Example: axilite\_read(AXILITE\_VVCT, 1, x"6000", “Read from Peripheral 1”); |

|  |
| --- |
| axilite\_check (VVCT, vvc\_instance\_idx, addr, data, msg, [alert\_level, [scope]]) |
| Example: axilite\_check(AXILITE\_VVCT, 1, x"6000", x”393B”, “Check data from Peripheral 1”); |

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

AXI4-Lite VVC Status record signal **´vvc\_status´ --** accessible via **shared\_axilite\_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 | AXILITE\_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”325A” | The address of a SW accessible register. Could be offset or full address depending on the DUT |
| data | std\_logic\_vector | x”F1A332D3” | The data to be written (in axilite\_write) or the expected data (in axilite\_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 | “AXILITE VVC” | A string describing the scope from which the log/alert originates. In a simple single sequencer typically  "AXILITE BFM". In a verification component typically "AXILITE VVC ". |

VVC entity signals

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

VVC entity generic constants

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** | |
| GC\_ADDR\_WIDTH | integer | 8 | Width of the AXI4-Lite address bus | |
| GC\_DATA\_WIDTH | integer | 32 | Width of the AXI4-Lite data bus | |
| GC\_INSTANCE\_IDX | natural | 1 | Instance number to assign the VVC | |
| GC\_AXILITE\_CONFIG | t\_axilite\_bfm\_config | C\_AXILITE\_BFM\_CONFIG\_DEFAULT | Configuration for the AXI4-Lite BFM, see AXI4-Lite 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** |
| **axilite\_write()** | **axilite\_write(VVC, instance\_idx, addr, data, [byte\_enable,] msg, [scope])**  The axilite\_write() VVC procedure adds a write command to the AXI4-Lite 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 AXI4-Lite BFM axilite\_write() procedure, described in the AXI4-Lite BFM QuickRef.  axilite\_write can be called with or without byte\_enable constant. When not set, byte\_enable is set to all ‘1’, indicating that all bytes are valid.  Examples:  axilite\_write(AXILITE\_VVCT, 1, x”0011A000”, x”F102”, “Writing data to Peripheral 1”, C\_SCOPE);  axilite\_write(AXILITE\_VVCT, 1, C\_ADDR\_PERIPHERAL\_1, x”F102”, b”11”, “Writing data to Peripheral 1”, C\_SCOPE);  axilite\_write(AXILITE\_VVCT, 1, C\_ADDR\_DMA, x”1155F102”, “Writing data to DMA”, C\_SCOPE); |
| **axilite\_read()** | **axilite\_read(VVC, instance\_idx, addr, msg, [scope])**  The axilite\_read() VVC procedure adds a read command to the AXI4-Lite 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 AXI4-Lite BFM axilite\_read() procedure, described in the AXI4-Lite BFM QuickRef.  The value read from the 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 with fetch\_result() below).  Examples:  axilite\_read(AXILITE\_VVCT, 1, x”00099555”, “Read from Peripheral 1” C\_SCOPE);  axilite\_read(AXILITE\_VVCT, 1, C\_ADDR\_IO, “Read from IO device” C\_SCOPE);  **Example with fetch\_result() call. Result is placed in v\_data**  variable v\_cmd\_idx : natural; -- Command index for the last read  variable v\_data : work.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from read  (…)  axilite\_read(AXILITE\_VVCT, 1, x“112252AA”, “Read from Peripheral 1”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(AXILITE\_VVCT, 1);  await\_completion(AXILITE\_VVCT,1, v\_cmd\_idx, 100 ns, "Wait for read to finish");  fetch\_result(AXILITE\_VVCT,1, v\_cmd\_idx, **v\_data**, "Fetching result from read operation"); |
| **axilite\_check()** | **axilite\_check(VVC, instance\_idx, addr, data, msg, [alert\_level, [scope]])**  The axilite\_check() VVC procedure adds a check command to the AXI4-Lite 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 AXI4-Lite BFM axilite\_check() procedure, described in the AXI4-Lite BFM QuickRef. The axilite\_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.  Example:  axilite\_check(AXILITE\_VVCT, 1, x”00099555”, x”393B”, “Check data from Peripheral 1”, ERROR, C\_SCOPE); |

# VVC Configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AXILITE\_BFM\_CONFIG\_DEFAULT** | **Description** | |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_AXILITE\_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 |
| bfm\_config | t\_axilite\_bfm\_config | C\_AXILITE\_BFM\_CONFIG\_DEFAULT | Configuration for AXI4-Lite BFM. See quick reference for AXI4-Lite BFM | |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | VVC dedicated message ID panel | |

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

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

shared\_axilite\_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\_axilite\_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 an activity watchdog which monitors VVC activity and will alert if no VVC activity is registered within a selected timeout value. The VVCs will register their presence to the activity watchdog at start-up, and report when busy and not, using dedicated activity watchdog methods and triggering the global\_trigger\_activity\_watchdog signal, during simulations.

Include activity\_watchdog(num\_exp\_vvc, timeout, alert\_level, msg) in the testbench to start using the activity watchdog.   
More information can be found in UVVM Essential Mechanisms PDF in the UVVM VVC Framework doc folder.

# VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t\_axilite\_if* in order to improve readability of the code. Since the AXI4-Lite 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, the could look like:

signal axilite\_if\_1 : t\_axilite\_if( write\_address\_channel( awaddr( C\_ADDR\_WIDTH -1 downto 0)),

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

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

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

read\_data\_channel ( rdata ( 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 AXI4-Lite standard, please see the AXI4-Lite specification “AMBA® AXI™ and ACE™ Protocol

Specification - AXI3™, AXI4™, and AXI4-Lite™ ACE and ACE-Lite™”, available from ARM.

# Compilation

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

* ***UVVM Utility Library (UVVM-Util), version 2.11.0 and up***
* ***UVVM VVC Framework, version 2.7.1 and up***
* ***AXI4-Lite BFM***
* ***Bitvis VIP Scoreboard***

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

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