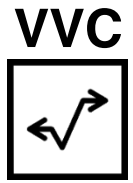
AXI4-Lite VVC – Quick Reference

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



axilite_vvc.vhd

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_read (VVCT, vvc_instance_idx, addr, [TO_SB,] msg, [scope])

Example: axilite_read(AXILITE_VVCT, 1, x"6000", "Read from Peripheral 1 and store data in VVC. Must be retrieved later using fetch result.");
axilite_read(AXILITE_VVCT, 1, x"600F", TO_SB, "Read from Peripheral and send result to scoreboard");

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**

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
force_single_pending_transaction	boolean	false
unwanted_activity_severity	t_alert_level	C_UNWANTED_ACTIVITY_SEVERITY

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

Common VVC procedures applicable for this VVC

- See UVVM Methods QuickRef for details.

await_completion() (wanted_idx parameter not supported)

enable_log_msg()

disable_log_msg()

fetch_result()

flush_command_queue()

terminate_current_command()

terminate_all_commands()

insert_delay()

get_last_received_cmd_idx()

VHDL 2008 only

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 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 [ ].

1 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 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-Lite procedures in axilite_channel_handler_pkg.vhd.
	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, [TO_SB,] msg, [scope])**
	
	The axilite_read() VVC procedure adds a read command to the AXI4-Lite 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-Lite procedures in axilite_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, but the read data will be stored in the VVC for a potential future fetch (see example with fetch_result() below).
	
	If the option TO_SB is applied the received data will be sent to the AXI Lite VVC dedicated scoreboard where it will be checked against the expected value (provided by the testbench).
	
	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 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-lite procedures in axilite_channel_handler_pkg.vhd. 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);
```

2 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. 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_axilite_vvc_config(1).inter_bfm_delay.delay_in_time := 50 ns;
shared_axilite_vvc_config(1).bfm_config.clock_period       := 10 ns;
```

3 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

4 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.

5 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-Lite transaction info record fields. Transaction type: t_base_transaction (BT) **-** accessible via **shared_axilite_vvc_transaction_info.bt_wr** and **shared_axilite_vvc_transaction_info.bt_rd***

Info field	Type	Default	Description
operation	t_operation	NO_OPERATION	Current VVC operation, e.g. INSERT_DELAY, POLL_UNTIL, READ, WRITE.
vvc_meta	t_vvc_meta	C_VVC_META_DEFAULT	VVC meta data of the executing VVC command.
→ msg	string	" "	Message of executing VVC command.
→ cmd_idx	integer	-1	Command index of executing VVC command.
transaction_status	t_transaction_status	C_TRANSACTION_STATUS_DEFAULT	Set to INACTIVE, IN_PROGRESS, FAILED or SUCCEEDED during a transaction.

*Table 5.2 AXI4-Lite transaction info record fields. Transaction type t_ax_transaction (ST) – accessible via **shared_axilite_vvc_transaction_info.st_aw** and **shared_axilite_vvc_transaction_info.st_ar***

Info field	Type	Default	Description
operation	t_operation	NO_OPERATION	Current VVC operation, e.g. INSERT_DELAY, POLL_UNTIL, READ, WRITE.
axaddr	unsigned(31 downto 0)	0x0	Address for a read or write transaction

vvc_meta	t_vvc_meta	C_VVC_META_DEFAULT	VVC meta data of the executing VVC command.
→ msg	string	" "	Message of executing VVC command.
→ cmd_idx	integer	-1	Command index of executing VVC command.
transaction_status	t_transaction_status	C_TRANSACTION_STATUS_DEFAULT	Set to INACTIVE, IN_PROGRESS, FAILED or SUCCEEDED during a transaction.

*Table 5.3 AXI4-Lite transaction info record fields. Transaction type t_w_transaction (ST) – accessible via **shared_axilite_vvc_transaction_info.st_w***

Info field	Type	Default	Description
operation	t_operation	NO_OPERATION	Current VVC operation, e.g. INSERT_DELAY, POLL_UNTIL, READ, WRITE.
wdata	std_logic_vector(255 downto 0)	0x0	Write data
wstrb	std_logic_vector(31 downto 0)	0x0	Write strobe
vvc_meta	t_vvc_meta	C_VVC_META_DEFAULT	VVC meta data of the executing VVC command.
→ msg	string	" "	Message of executing VVC command.
→ cmd_idx	integer	-1	Command index of executing VVC command.
transaction_status	t_transaction_status	C_TRANSACTION_STATUS_DEFAULT	Set to INACTIVE, IN_PROGRESS, FAILED or SUCCEEDED during a transaction.

*Table 5.4 AXI4-Lite transaction info record fields. Transaction type t_b_transaction (ST) – accessible via **shared_axilite_vvc_transaction_info.st_b***

Info field	Type	Default	Description
operation	t_operation	NO_OPERATION	Current VVC operation, e.g. INSERT_DELAY, POLL_UNTIL, READ, WRITE.
vvc_meta	t_vvc_meta	C_VVC_META_DEFAULT	VVC meta data of the executing VVC command.
→ msg	string	" "	Message of executing VVC command.
→ cmd_idx	integer	-1	Command index of executing VVC command.
transaction_status	t_transaction_status	C_TRANSACTION_STATUS_DEFAULT	Set to INACTIVE, IN_PROGRESS, FAILED or SUCCEEDED during a transaction.

*Table 5.5 AXI4-Lite transaction info record fields. Transaction type t_r_transaction (ST) – accessible via **shared_axilite_vvc_transaction_info.st_r***

Info field	Type	Default	Description
operation	t_operation	NO_OPERATION	Current VVC operation, e.g. INSERT_DELAY, POLL_UNTIL, READ, WRITE.
rdata	std_logic_vector(255 downto 0)	0x0	Read data
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.

6 Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the `TO_SB` parameter in supported method calls, i.e. `axilite_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 AXI Lite VVC scoreboard is per default 256 bits wide std logic vector. When sending expected data to the scoreboard, where the data width is smaller than the default scoreboard width, we recommend zero-padding the data with the pad_axilite_sb() function. E.g. `AXILITE_VVC_SB.add_expected(<AXI Lite VVC instance number>, pad_axilite_sb(<exp_data>));`

See the Generic Scoreboard Quick Reference PDF in the Bitvis VIP Scoreboard document folder for a complete list of available commands and additional information. The AXI4-Lite VVC scoreboard is accessible from the testbench as a shared variable `AXILITE_VVC_SB`, located in the `vvc_methods_pkg.vhd`. All of the listed Generic Scoreboard commands are available for the AXI4-Lite VVC scoreboard using this shared variable.

7 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)) );
```

8 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. AXI-Lite read/check 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_axilite_vvc_config(C_VVC_INDEX).unwanted_activity_severity := NO_ALERT;
```

Note that the ready signals (`awready`, `wready`, `arready`) are not monitored in this VVC. The ready signals are allowed to be set independently of the valid signals (`awvalid`, `wvalid`, `arvalid`), and there is no method to differentiate between the unwanted activity and intended activity. See the AXI-Lite protocol specification for more information.

The unwanted activity detection is ignored when the valid signals (`bvalid`, `rvalid`) go 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 AXI-Lite VVC, the unwanted activity detection feature is enabled (`unwanted_activity_severity := ERROR`) by default.

9 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.

10 Compilation

AXI4-Lite 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***
- ***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_channel_handler_pkg.vhd	Package containing procedures for accessing AXI4-Lite channels. Only for use by the VVC
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
bitvis_vip_axilite	vvc_context.vhd	AXI4-Lite VVC context

11 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 UVVM at info@uvvm.org