AXI4 VVC – Quick Reference

For general information see UVVM VVC Framework Essential Mechanisms located in uvvm_vvc_framework/doc.

NOTE: As of UVVM v3.x, all shared variables have been made protected. This means that any access to shared variables must be done using get- and set-methods. This documentation has not yet been updated with the methods for accessing these variables, but will be very soon. Please refer to section 2 of Avalon_mm_vvc_QuickRef for example usage of protected shared variables

VVC

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");
```


VHDL 2008 only

UVVM™

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**

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
force_single_pending_transaction	boolean	false

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

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()

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
awid	std_logic_vector	x"01"	Identification tag for a write transaction
awaddr	unsigned	x"325A"	The address of the first transfer in a write transaction
awlen	unsigned(7 downto 0)	x"01"	The number of data transfers in a write transaction
awsize	Integer range 1 to 128	4	The number of bytes in each data transfer in a write transaction (Must be a power of two)
awburst	t_axburst	INCR	Burst type, indicates how address changes between each transfer in a write transaction
awlock	t_axlock	NORMAL	Provides information about the atomic characteristics of a write transaction
awcache	std_logic_vector(3 downto 0)	"0000"	Indicates how a write transaction is required to progress through a system
awprot	t_axprot	UNPRIVILEGED_UNSECURE_DATA	Protection attributes of a write transaction. Privilege, security level and access type
awqos	std_logic_vector(3 downto 0)	"0000"	Quality of Service identifier for a write transaction
awregion	std_logic_vector(3 downto 0)	"0000"	Region indicator for a write transaction
awuser	std_logic_vector	x"01"	User-defined extension for the 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
arid	std_logic_vector	x"01"	Identification tag for a read transaction
araddr	unsigned	x"325A"	The address of the first transfer in a read transaction
arlen	unsigned(7 downto 0)	x"01"	The number of data transfers in a read transaction
arsize	Integer range 1 to 128	4	The number of bytes in each data transfer in a read transaction (Must be a power of two)
arburst	t_axburst	INCR	Burst type, indicates how address changes between each transfer in a read transaction
arlock	t_axlock	NORMAL	Provides information about the atomic characteristics of a read transaction
arcache	std_logic_vector(3 downto 0)	"0000"	Indicates how a read transaction is required to progress through a system
arprot	t_axprot	UNPRIVILEGED_UNSECURE_DATA	Protection attributes of a read transaction. Privilege, security level and access type
arqos	std_logic_vector(3 downto 0)	"0000"	Quality of Service identifier for a read transaction
arregion	std_logic_vector(3 downto 0)	"0000"	Region indicator for a read transaction
aruser	std_logic_vector	x"01"	User-defined extension for the read address 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 [ ].

1 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");
```

2 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;
```

3 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

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 transaction info record fields. Transaction type: t_base_transaction (BT) **-** accessible via **shared_axi_vvc_transaction_info.bt_wr** and **shared_axi_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.
axid	std_logic_vector(31 downto 0)		Identification tag for a read or write transaction
axaddr	unsigned(31 downto 0)	0x0	Address for a read or write transaction
axlen	unsigned(7 downto 0)	0x0	Burst length for a read or write transaction
axsize	integer range 1 to 128	4	Burst size for a read or write transaction
axburst	t_axburst	INCR	Burst type for a read or write transaction
axlock	t_axlock	NORMAL	Lock value for a read or write transaction
axcache	std_logic_vector(3 downto 0)	0x0	Cache value for a read or write transaction
axprot	t_axprot	UNPRIVILEGED_NONSECURE_DATA	Protection value for a read or write transaction
axqos	std_logic_vector(3 downto 0)	0x0	QOS value for a read or write transaction
axregion	std_logic_vector(3 downto 0)	0x0	Region value for a read or write transaction
axuser	std_logic_vector(127 downto 0)	0x0	User value 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	t_slv_array(0 to 255)(255 downto 0)	0x0	Write data
wstrb	t_slv_array(0 to 255)(31 downto 0)	0x0	Write strobe
wuser	t_slv_array(0 to 255)(127 downto 0)	0x0	User value
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.
bid	std_logic_vector(31 downto 0)	0x0	Identification tag
bresp	t_xresp	OKAY	Write response
buser	std_logic_vector(127 downto 0)	0x0	User value for write response channel
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.
rid	std_logic_vector(31 downto 0)	0x0	Identification tag for read data channel
rdata	t_slv_array(0 to 255)(255 downto 0)	0x0	Read data array
rresp	t_xresp_array(0 to 255)	OKAY	Read response array
ruser	t_slv_array(0 to 255)(127 downto 0)	0x0	Read user extension array
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 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.

7 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)));
```

8 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.

9 Compilation

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

- ***UVVM Utility Library (UVVM-Util), version 2.16.0 and up***
- ***UVVM VVC Framework, version 2.12.0 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	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 to support out of order read data
bitvis_vip_axi	axi_channel_handler_pkg.vhd	Package containing procedures for accessing AXI4 channels. Only for use by the VVC
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

10 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