Avalon-MM VVC – Quick Reference

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

VVC

avalon_mm_vvc.vhd

avalon_mm_write (VVCT, vvc_instance_idx, addr, data, [byte_enable,] msg, [scope])

Example: avalon_mm_write(AVALON_MM_VVCT, 1, x"00006000", x"AABBF102", "Writing to Peripheral 1");

avalon_mm_read (VVCT, vvc_instance_idx, addr, msg, [scope])

Example: avalon_mm_read(AVALON_MM_VVCT, 1, x"10056000", "Reading from Peripheral 1");

avalon_mm_check (VVCT, vvc_instance_idx, addr, data, msg, [alert_level, [scope]])

Example: avalon_mm_check(AVALON_MM_VVCT, 1, x"FF113000", x"0000393B", "Check data from Peripheral 1");

avalon_mm_reset (VVCT, vvc_instance_idx, num_rst_cycles, msg, [scope])

Example: avalon_mm_reset(AVALON_MM_VVCT, 1, 5, "Resetting Avalon-MM interface for 5 cycles");

avalon_mm_lock (VVCT, vvc_instance_idx, msg, [scope])

Example: avalon_mm_lock(AVALON_MM_VVCT, 1, "Locking Avalon MM Interface");

avalon_mm_unlock (VVCT, vvc_instance_idx, msg, [scope])

Example: avalon_mm_unlock(AVALON_MM_VVCT, 1, "Unlocking Avalon MM Interface");

Avalon-MM VVC Configuration record ´**vvc_config**´ - accessible via **shared_avalon_mm_vvc_config**

Record element	Type	C_AVALON_MM_VVC_CONFIG_DEFAULT
inter_bfm_delay	t_inter_bfm_delay	C_AVALON_MM_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
use_read_pipeline	boolean	true
num_pipeline_stages	natural	5
bfm_config	t_avalon_mm_bfm_config	C_AVALON_MM_BFM_CONFIG_DEFAULT
msg_id_panel	t_msg_id_panel	C_VVC_MSG_ID_PANEL_DEFAULT

Avalon-MM VVC Status record signal ´**vvc_status**´ - accessible via **shared_avalon_mm_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()

enable_log_msg()

disable_log_msg()

fetch_result()

flush_command_queue()

terminate_current_command()

terminate_all_commands()

insert_delay()

get_last_received_cmd_idx()

UVVM™
VHDL 2008 only

VVC target parameters

Name	Type	Example(s)	Description
VVCT	t_vvc_target_record	AVALON_MM_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"0000325A"	The address of a Avalon-MM accessible register. Could be offset or full address depending on the DUT
data	std_logic_vector	x"F1A332D3"	The data to be written (in avalon_mm_write) or the expected data (in avalon_mm_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	"AVALON MM VVC"	A string describing the scope from which the log/alert originates. In a simple single sequencer typically "AVALON MM BFM". In a verification component typically "AVALON MM VVC ".

VVC entity signals

Name	Type	Description
clk	std_logic	VVC Clock signal
avalon_mm_vvc_master_if	t_avalon_mm_if	See Avalon-MM BFM documentation

VVC entity generic constants

Name	Type	Default	Description
GC_ADDR_WIDTH	integer	8	Width of the Avalon-MM address bus
GC_DATA_WIDTH	integer	32	Width of the Avalon-MM data bus
GC_INSTANCE_IDX	natural	1	Instance number to assign the VVC
GC_AVALON_MM_CONFIG	t_avalon_mm_bfm_config	C_AVALON_MM_BFM_CONFIG_DEFAULT	Configuration for the Avalon-MM BFM, see Avalon-MM 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

avalon_mm_write()	**avalon_mm_write(VVCT, vvc_instance_idx, addr, data, [byte_enable,] msg, [scope])**
	
	The avalon_mm_write() VVC procedure adds a write command to the Avalon-MM 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 Avalon-MM BFM avalon_mm_write() procedure, described in the Avalon-MM BFM QuickRef.
	avalon_mm_write can be called with or without byte_enable constant. When not set, byte_enable is interpreted as all '1', indicating that all bytes are valid.
	
	Examples:
	`avalon_mm_write(AVALON_MM_VVCT, 1, x"11221100", x"0000F102", "Writing to Peripheral 1", C_SCOPE);`
	`avalon_mm_write(AVALON_MM_VVCT, 1, C_ADDR_DMA, x"F102", "1111", "Writing to DMA", C_SCOPE);`

avalon_mm_read()	**avalon_mm_read(VVCT, vvc_instance_idx, addr, msg, [scope])**
	
	The avalon_mm_read() VVC procedure adds a read command to the Avalon-MM 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 Avalon-MM BFM avalon_mm_read() procedure, described in the Avalon-MM BFM QuickRef.
	The value read from 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 below).
	
	Using read pipeline:
	If vvc_config.use_read_pipeline has been set to true, the VVC will perform the read transaction using the BFM procedures avalon_mm_read_request and avalon_mm_read_response. First, the VVC executor will check if the number of pending commands in the pipeline will exceed the number of pipeline stages. If this is the case, the VVC executor will stall the read transaction until a command in the pipeline has been executed. The command executor will then let the BFM start the read request. After the read request has completed, the command will be added to the command response queue, which will run the BFM procedure avalon_mm_read_response.
	
	<u>Example with fetch_result() call (Result is placed in v_data)</u>
	`variable v_cmd_idx : natural;`
	`variable v_data    : bitvis_vip_avalon_mm.vvc_cmd_pkg.t_vvc_result;`
	`(…)`
	`avalon_mm_read(AVALON_MM_VVCT, 1, x"112252AA", "Read from Peripheral 1", C_SCOPE);`
	`v_cmd_idx := get_last_received_cmd_idx(AVALON_MM_VVCT, 1); -- Store the command index (integer) for the last read`
	`await_completion(AVALON_MM_VVCT,1, v_cmd_idx, 100 ns, "Wait for read to finish");`
	`fetch_result(AVALON_MM_VVCT,1, v_cmd_idx, v_data, "Fetching result from read operation");`

avalon_mm_check()

avalon_mm_check(VVCT, vvc_instance_idx, addr, data, msg, [alert_level, [scope]])

The avalon_mm_check() VVC procedure adds a check command to the Avalon-MM 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 Avalon-MM BFM avalon_mm_check() procedure, described in the Avalon-MM BFM QuickRef. The avalon_mm_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.

Using read pipeline:
If vvc_config.use_read_pipeline has been set to true, the VVC will perform the check transaction using the BFM procedures avalon_mm_read_request and avalon_mm_check_response, similar to the procedure described in avalon_mm_read.

Example:
```
    avalon_mm_check(AVALON_MM_VVCT, 1, x"11A49800", x"0000393B", "Check data from Peripheral 1", ERROR, C_SCOPE);
```

avalon_mm_reset()

avalon_mm_reset(VVCT, vvc_instance_idx, num_rst_cycles, msg, [scope])

The avalon_mm_reset() VVC procedure adds a reset command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the reset command is scheduled to run, the executor calls the Avalon-MM BFM avalon_mm_reset() procedure, described in the Avalon-MM BFM QuickRef.

Example:
```
    avalon_mm_reset(AVALON_MM_VVCT, 1, 5, "Resetting Avalon MM Interface", C_SCOPE);
```

avalon_mm_lock()

avalon_mm_lock(VVCT, vvc_instance_idx, msg, [scope])

The avalon_mm_lock() VVC procedure adds a lock command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the lock command is scheduled to run, the executor calls the Avalon-MM BFM avalon_mm_lock() procedure, described in the Avalon-MM BFM QuickRef.

Example:
```
    avalon_mm_lock(AVALON_MM_VVCT, 1, "Locking Avalon MM Interface", C_SCOPE);
```

avalon_mm_unlock()

avalon_mm_unlock(VVCT, vvc_instance_idx, msg, [scope])

The avalon_mm_unlock() VVC procedure adds an unlock command to the Avalon-MM VVC executor queue, which will run as soon as all preceding commands have completed. When the lock command is scheduled to run, the executor calls the Avalon-MM BFM avalon_mm_unlock() procedure, described in the Avalon-MM BFM QuickRef.

Example:
```
    avalon_mm_unlock(AVALON_MM_VVCT, 1, "Locking Avalon MM Interface", C_SCOPE);
```

2 VVC Configuration

Record element	Type	C_AVALON_MM_BFM_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_AVALON_MM_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
num_pipeline_stages	natural	5	Max read_requests in pipeline
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_avalon_mm_vvc_config(1).inter_bfm_delay.delay_in_time := 50 ns;
  shared_avalon_mm_vvc_config(1).bfm_config.use_waitrequest    := true;
```

3 VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared_avalon_mm_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 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.

5 VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t_avalon_mm_if* in order to improve readability of the code. Since the Avalon-MM 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 avalon_mm_if_1 : t_avalon_mm_if( address(C_ADDR_WIDTH-1 downto 0),
                                        byte_enable((C_DATA_WIDTH/8)-1 downto 0),
                                        writedata(C_DATA_WIDTH-1 downto 0),
                                        readdata(C_DATA_WIDTH-1 downto 0) );
```

6 Additional Documentation

Additional documentation about UVVM and its features can be found under “uvvm_vvc_framework/doc/”.
For additional documentation on the Avalon-MM standard, please see the Avalon specification “Avalon Interface Specifications, MNL-AVABUSREF”, available from Altera.

7 Compilation

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

- ***UVVM Utility Library (UVVM-Util), version 2.12.0 and up***
- ***UVVM VVC Framework, version 2.7.3 and up***
- ***Avalon-MM BFM***
- ***Bitvis VIP Scoreboard***

Before compiling the Avalon-MM VVC, assure that uvvm_vvc_framework, uvvm_util and bitvis_vip_scoreboard have been compiled.

See the UVVM Essential Mechanisms located in uvvm_vvc_framework/doc for information about compile scripts.

Compile order for the Avalon-MM VVC:

Compile to library	File	Comment
bitvis_vip_avalon_mm	avalon_mm_bfm_pkg.vhd	Avalon-MM BFM
bitvis_vip_avalon_mm	transaction_pkg.vhd	Avalon-MM transaction package with DTT types, constants etc.
bitvis_vip_avalon_mm	vvc_cmd_pkg.vhd	Avalon-MM VVC command types and operations
bitvis_vip_avalon_mm	../uvvm_vvc_framework/src_target_dependent/td_target_support_pkg.vhd	UVVM VVC target support package, compiled into the Avalon-MM VVC library.
bitvis_vip_avalon_mm	../uvvm_vvc_framework/src_target_dependent/td_vvc_framework_common_methods_pkg.vhd	UVVM VVC framework common methods compiled into the Avalon-MM VVC library
bitvis_vip_avalon_mm	vvc_methods_pkg.vhd	Avalon-MM VVC methods
bitvis_vip_avalon_mm	../uvvm_vvc_framework/src_target_dependent/td_queue_pkg.vhd	UVVM queue package for the VVC
bitvis_vip_avalon_mm	../uvvm_vvc_framework/src_target_dependent/td_vvc_entity_support_pkg.vhd	UVVM VVC entity support compiled into the Avalon-MM VVC library
bitvis_vip_avalon_mm	avalon_mm_vvc.vhd	Avalon-MM VVC

8 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 Avalon-MM. The given VIP complies with the basic Avalon-MM protocol and thus allows a normal access towards an Avalon-MM interface. This VIP is not an Avalon-MM protocol checker. For a more advanced VIP please contact Bitvis AS at support@bitvis.no