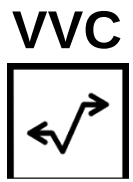
Avalon-Stream VVC – Quick Reference

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



avalon_st_vvc.vhd

Avalon-Stream Master

In order to use the Avalon-Stream VVC in master mode, it must be instantiated in the test harness by setting the generic constant *'GC_MASTER_MODE'* to TRUE.

avalon_st_transmit (VVCT, vvc_instance_idx, [channel_value], data_array, msg, [scope])

Example: avalon_st_transmit(AVALON_ST_VVCT, 0, v_channel, v_data_array(0 to v_numWords-1), "Send v_numWords on v_channel to DUT", C_SCOPE);
avalon_st_transmit(AVALON_ST_VVCT, 0, (x"01", x"02", x"03", x"04"), "Send 4 bytes to DUT");

Note that this procedure can only be called when the AVALON_ST VVC is instantiated in master mode, i.e. setting the generic constant 'GC_MASTER_MODE' to true.

Avalon-Stream Slave

In order to use the Avalon-Stream VVC in slave mode, it must be instantiated in the test harness by setting the generic constant *'GC_MASTER_MODE'* to FALSE.

avalon_st_receive (VVCT, vvc_instance_idx, data_array_len, data_word_size, msg, [scope])

Example: avalon_st_receive(AVALON_ST_VVCT, 1, v_data_array'length, v_data_array(0)'length, "Avalon ST Receive: Receive data will be stored in VVC. Retrieve later using fetch result() ");
avalon_st_receive(AVALON_ST_VVCT, 1, v_data_array'length, v_data_array(0)'length, " Avalon ST Receive: Receive data will be sent to scoreboard ");

avalon_st_expect (VVCT, vvc_instance_idx, [channel_exp], data_exp, msg, [alert_level, [scope]])

Example: avalon_st_expect(AVALON_ST_VVCT, 1, v_channel, v_data_array(0 to v_numWords-1), "Expect v_numWords on v_channel", ERROR, C_SCOPE);
avalon_st_expect(AVALON_ST_VVCT, 1, (x"01", x"02", x"03", x"04"), "Expect 4 bytes");

Copyright © 2019 by Bitvis AS. All rights reserved.

VHDL 2008 only

Avalon-Stream VVC Configuration record ´**vvc_config**´ **--** accessible via **shared_avalon_st_vvc_config**

Record element	Type	C_AVALON_ST_VVC_CONFIG_DEFAULT
inter_bfm_delay	t_inter_bfm_delay	C_AVALON_ST_INTER_BFM_DELAY_DEFAULT
cmd_queue_count_max	natural	C_CMD_QUEUE_COUNT_MAX
cmd_queue_count_threshold	natural	C_CMD_QUEUE_COUNT_THRESHOLD
cmd_queue_count_threshold_severity	t_alert_level	C_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY
result_queue_count_max	natural	C_RESULT_QUEUE_COUNT_MAX
result_queue_count_threshold	natural	C_RESULT_QUEUE_COUNT_THRESHOLD
result_queue_count_threshold_severity	t_alert_level	C_RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY
bfm_config	t_avalon_st_bfm_config	C_AVALON_ST_BFM_CONFIG_DEFAULT
msg_id_panel	t_msg_id_panel	C_VVC_MSG_ID_PANEL_DEFAULT

Avalon-Stream VVC Status record signal ´**vvc_status**´ **--** accessible via **shared_avalon_st_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_[any]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()

VVC target parameters

Name	Type	Example(s)	Description
VVCT	t_vvc_target_record	AVALON_ST_VVCT	VVC target type compiled into each VVC in order to differentiate between VVCs.
vvc_instance_idx	integer	0	Instance number of the VVC

VVC functional parameters

Name	Type	Example(s)	Description
channel_value channel_exp	std_logic_vector	x"01"	Channel number for the data being transferred or expected. The value is limited by max_channel in the bfm_config.

data_array
data_exp
	t_slv_array	(x"D0D1", x"D2D3")	An array of SLVs containing the data to be sent/received.
			data_array(0) is sent/received first, while data_array(data_array'high) is sent/received last.
			For clarity, data_array is required to be ascending, for example defined by the test sequencer as follows:
			`variable v_data_array : t_slv_array(0 to C_MAX_WORDS-1)(C_MAX_WORD_LENGTH-1 downto 0);`
			For simplicity, the word_length can only be the size of the configured symbol (usually with packet-based transfers) or the size of the data bus (usually with data-based transfers).
			`variable v_data_array : t_slv_array(0 to C_MAX_WORDS-1)(C_SYMBOL_WIDTH-1 downto 0);`
			`variable v_data_array : t_slv_array(0 to C_MAX_WORDS-1)(C_DATA_BUS_LENGTH-1 downto 0);`

data_array_len	natural	20	Length of the data_array expected to be received (number of words).
data_word_size	natural	8	Size of the data words in the data_array expected to be received.
alert_level	t_alert_level	ERROR or TB_WARNING	Set the severity for the alert that may be asserted by the procedure.
msg	string	"Send data"	A custom message to be appended in the log/alert
scope	string	"AVALON_ST_VVC"	A string describing the scope from which the log/alert originates. In a simple single sequencer typically "AVALON_ST_BFM". In a verification component typically "AVALON_ST_VVC ".

VVC entity signals

Name	Type	Description
clk	std_logic	VVC Clock signal
avalon_st_vvc_if	t_avalon_st_if	See Avalon-Stream BFM documentation

VVC entity generic constants

Name	Type	Default	Description
GC_VVC_IS_MASTER	boolean	-	Set to true when this VVC instance is an Avalon-Stream master (data is output from BFM). Set to false when this VVC is an Avalon-Stream slave (data is input to BFM.)
GC_CHANNEL_WIDTH	integer	1	Width of the Avalon-Stream channel signal. *Note 1*: if CHANNEL is wider than 8, increase the value of the constant C_AVALON_ST_CHANNEL_MAX_LENGTH in the local_adaptations_pkg. *Note 2*: If the CHANNEL signal is not used, refer to description in Section 5.
GC_DATA_WIDTH	integer	-	Width of the Avalon-Stream data bus. *Note*: if DATA is wider than 512, increase the value of the constant C_AVALON_ST_WORD_MAX_LENGTH in the local_adaptations_pkg.
GC_DATA_ERROR_WIDTH	integer	1	Width of the Avalon-Stream data error signal. *Note*: If the DATA_ERROR signal is not used, refer to description in Section 5.
GC_EMPTY_WIDTH	integer	1	Width of the Avalon-Stream empty signal. *Note*: If the EMPTY signal is not used, refer to description in Section 5.
GC_INSTANCE_IDX	natural	-	Instance number to assign the VVC.
GC_AVALON_ST_BFM_CONFIG	t_avalon_st_bfm_config	C_AVALON_ST_BFM_CONFIG_DEFAULT	Configuration for the Avalon-Stream BFM, see Avalon-Stream 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

Procedure	Description
avalon_st_transmit()	**avalon_st_transmit (VVCT, vvc_instance_idx, channel_value, data_array, msg, [scope])** The avalon_st_transmit() VVC procedure adds a transmit command to the Avalon-Stream VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the Avalon-Stream BFM avalon_st_transmit() procedure, described in the Avalon-Stream BFM QuickRef. The avalon_transmit() procedure can only be called when the AVALON VVC is instantiated in master mode, i.e. setting the generic constant 'GC_MASTER_MODE' to true.

avalon_st_receive()	**avalon_st_receive (VVCT, vvc_instance_idx, data_array_len, data_word_size, msg, [scope])**
	
	The avalon_st_receive() VVC procedure adds a receive command to the Avalon-Stream VVC executor queue, which will run as soon as all preceding commands have completed.
	When the command is scheduled to run, the executor calls the Avalon-Stream BFM avalon_st_receive() procedure, described in the Avalon-Stream BFM QuickRef.
	The avalon_receive() procedure can only be called when the AVALON VVC is instantiated in slave mode, i.e. setting the generic constant 'GC_MASTER_MODE' to false.
	The value received from the DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the received data and metadata will be stored in the VVC for a potential future fetch (see example with *fetch_result* below).
	
	Example with fetch_result() call: Result is placed in **`v_result`**
	`variable v_cmd_idx : natural;  -- Command index for the last receive`
	`variable v_result  : work.vvc_cmd_pkg.t_vvc_result; -- Result from receive (data and metadata)`
	`(…)`
	`avalon_st_receive(AVALON_ST_VVCT, 1, v_data_array'length, v_data_array(0)'length, "Receive data in VVC");`
	`v_cmd_idx := get_last_received_cmd_idx(AVALON_ST_VVCT, 1);`
	`await_completion(AVALON_ST_VVCT, 1, 1 ms, "Wait for receive to finish");`
	`fetch_result(AVALON_ST_VVCT, 1, v_cmd_idx, v_result, "Fetching result from receive operation");`

avalon_st_expect()	**avalon_st_expect (VVCT, vvc_instance_idx, channel_exp, data_exp, msg, [alert_level, [scope]])** The avalon_st_expect() VVC procedure adds an expect command to the Avalon-Stream VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the Avalon-Stream BFM avalon_st_expect() procedure, described in the Avalon-Stream BFM QuickRef. The avalon_expect() procedure can only be called when the AVALON VVC is instantiated in slave mode, i.e. setting the generic constant 'GC_MASTER_MODE' to false.

2 VVC Instantiation

In order to select between the master and slave modes, the VVC must be instantiated using the correct value of the generic constant GC_VVC_IS_MASTER in the testbench or test-harness. Example instantiations of the VVC in both operation supplied for ease of reference.

Mode	Instantiation

Master	`i_avalon_st_vvc_master : entity work.avalon_st_vvc`
	`  generic map(`
	`  GC_VVC_IS_MASTER => true,`
	`  GC_CHANNEL_WIDTH => GC_CHANNEL_WIDTH,`
	`  GC_DATA_WIDTH => GC_DATA_WIDTH,`
	`  GC_DATA_ERROR_WIDTH => GC_ERROR_WIDTH,`
	`  GC_EMPTY_WIDTH => GC_EMPTY_WIDTH,`
	`  GC_INSTANCE_IDX => 0)`
	`  port map(`
	`  clk => clk,`
	`  avalon_st_vvc_if => avalon_st_master_if);`

Mode	Instantiation

Slave	`i_avalon_st_vvc_slave : entity work.avalon_st_vvc`
	`  generic map(`
	`  GC_VVC_IS_MASTER => false,`
	`  GC_CHANNEL_WIDTH => GC_CHANNEL_WIDTH,`
	`  GC_DATA_WIDTH => GC_DATA_WIDTH,`
	`  GC_DATA_ERROR_WIDTH => GC_ERROR_WIDTH,`
	`  GC_EMPTY_WIDTH => GC_EMPTY_WIDTH,`
	`  GC_INSTANCE_IDX => 1)`
	`  port map(`
	`  clk => clk,`
	`  avalon_st_vvc_if => avalon_st_slave_if);`

3 VVC Configuration

Record element	Type	C_AVALON_ST_VVC_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_AVALON_ST_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_CMD_QUEUE_COUNT_MAX	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_avalon_st_bfm_config	C_AVALON_ST_BFM_CONFIG_DEFAULT	Configuration for Avalon-Stream BFM. See quick reference for Avalon-Stream 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_avalon_st_vvc_config(1).inter_bfm_delay.delay_in_time := 50 ns;
shared_avalon_st_vvc_config(1).bfm_config.clock_period := 10 ns;
```

4 VVC Status

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

5 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.

6 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 6.1 Avalon Stream transaction info record fields. Transaction type: base transaction (BT) - accessible via **shared_avalon_st_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.
channel_value	slv(7 downto 0)	0x0	Channel number for the data being transferred or expected. The value is limited by max_channel in the bfm_config. The width of channel_value can be configured through the local_adaptations_pkg by changing the value of C_AVALON_ST_CHANNEL_MAX_LENGTH. Default value is 8.
data_array	t_slv_array(0 to 1024)(512 downto 0)	(others => (others => '0'))	An array of SLVs containing the data to be sent/received. data_array(0) is sent/received first, while data_array(data_array'high) is sent/received last. The length of the data words, as well as the maximum amount of data words in data_array, are configurable through the constants AVALON_ST_WORD_MAX_LENGTH and AVALON_ST_DATA_MAX_WORDS found in local_adaptations_pkg.
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.

7 VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t_avalon_st_if* in order to improve readability of the code. Since the Avalon-Stream interface buses 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_st_if : t_avalon_st_if (
  channel(C_CHANNEL_WIDTH-1 downto 0),
  data(C_DATA_WIDTH-1 downto 0),
  data_error(C_ERROR_WIDTH-1 downto 0),
  empty(log2(C_DATA_WIDTH/C_SYMBOL_WIDTH)-1 downto 0));
```

The widths of *channel, data_error* and *empty* are declared even when not used or connected to DUT.
Set the widths of unused signals to 1, for example `C_CHANNEL_WIDTH = 1`.

8 Additional Documentation

Additional documentation about UVVM and its features can be found under “/uvvm_vvc_framework/doc/”.

For additional documentation on the Avalon-Stream standard, refer to “Avalon® Interface Specifications, Chapter: Avalon Streaming Interfaces”, document number MNL-AVABUSREF, available from Intel.

9 Compilation

Avalon-Stream 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***
- ***Avalon-Stream BFM***

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

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

Compile order for the Avalon-Stream VVC:

Compile to library	File	Comment
bitvis_vip_avalon_st	avalon_st_bfm_pkg.vhd	Avalon-Stream BFM
bitvis_vip_avalon_st	local_adaptations_pkg.vhd	Avalon-Stream adaptations package for local modifications.
bitvis_vip_avalon_st	transaction_pkg.vhd	Avalon-Stream transaction package with DTT types, constants etc.
bitvis_vip_avalon_st	vvc_cmd_pkg.vhd	Avalon-Stream VVC command types and operations
bitvis_vip_avalon_st	../uvvm_vvc_framework/src_target_dependent/td_target_support_pkg.vhd	UVVM VVC target support package, compiled into the Avalon-Stream VVC library.
bitvis_vip_avalon_st	../uvvm_vvc_framework/src_target_dependent/td_vvc_framework_common_methods_pkg.vhd	UVVM framework common methods compiled into the Avalon-Stream VVC library
bitvis_vip_avalon_st	vvc_methods_pkg.vhd	Avalon-Stream VVC methods
bitvis_vip_avalon_st	../uvvm_vvc_framework/src_target_dependent/td_queue_pkg.vhd	UVVM queue package for the VVC
bitvis_vip_avalon_st	../uvvm_vvc_framework/src_target_dependent/td_vvc_entity_support_pkg.vhd	UVVM VVC entity support compiled into the Avalon-Stream VVC library
bitvis_vip_avalon_st	avalon_st_vvc.vhd	Avalon-Stream VVC

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

INTELLECTUAL PROPERTY

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.