

# UART VVC – Quick Reference

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

**uart\_transmit (VVCT, vvc\_instance\_idx, channel, data | {num\_words, randomisation}, msg, [scope])**

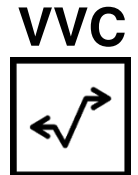
**Examples:** `uart_transmit(UART_VVCT, 1, TX, x"AF", "Sending data to DUT UART instance 1");`  
`uart_transmit(UART_VVCT, 1, TX, 5, RANDOM, "Sending 5 random bytes to DUT UART instance 1");`

**uart\_receive (VVCT, vvc\_instance\_idx, channel, [TO\_SB,] msg, [alert\_level, [scope]])**

**Example:** `uart_receive(UART_VVCT, 1, RX, "Receive data from DUT UART instance 1");`  
`uart_receive (UART_VVCT, 1, RX, TO_SB, "Receiving data from DUT UART instance 1 and passing on to SB", ERROR, C_SCOPE);`

**uart\_expect (VVCT, vvc\_instance\_idx, channel, data, msg, [max\_receptions, [timeout, [alert\_level, [scope]]]])**

**Example:** `uart_expect(UART_VVCT, 1, RX, x"42", "Expect data from DUT UART instance 1");`



*uart\_vvc.vhd  
uart\_rx\_vvc.vhd  
uart\_tx\_vvc.vhd*

UART VVC Configuration record `'vvc_config'` -- accessible via **shared\_uart\_vvc\_config**

Record element	Type	C_UART_VVC_CONFIG_DEFAULT
<code>inter_bfm_delay</code>	<code>t_inter_bfm_delay</code>	<code>C_UART_INTER_BFM_DELAY_DEFAULT</code>
<code>[cmd/result]_queue_count_max</code>	<code>natural</code>	<code>C_[CMD/RESULT]_QUEUE_COUNT_MAX</code>
<code>[cmd/result]_queue_count_threshold</code>	<code>natural</code>	<code>C_[CMD/RESULT]_QUEUE_COUNT_THRESHOLD</code>
<code>[cmd/result]_queue_count_threshold_severity</code>	<code>t_alert_level</code>	<code>C_[CMD/RESULT]_QUEUE_COUNT_THRESHOLD_SEVERITY</code>
<code>bfm_config</code>	<code>t_uart_bfm_config</code>	<code>C_UART_BFM_CONFIG_DEFAULT</code>
<code>error_injection</code>	<code>t_vvc_error_injection</code>	<code>C_ERROR_INJECTION_INACTIVE</code>
<code>bit_rate_checker</code>	<code>t_bit_rate_checker</code>	<code>C_BIT_RATE_CHECKER_DEFAULT</code>
<code>msg_id_panel</code>	<code>t_msg_id_panel</code>	<code>C_VVC_MSG_ID_PANEL_DEFAULT</code>

UART VVC Status record signal `'vvc_status'` -- accessible via **shared\_uart\_vvc\_status**

Record element	Type
<code>current_cmd_idx</code>	<code>natural</code>
<code>previous_cmd_idx</code>	<code>natural</code>
<code>pending_cmd_cnt</code>	<code>natural</code>

## 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	UART_VVCT	VVC target type compiled into each VVC in order to differentiate between VVCs.
vvc_instance_idx	integer	1	Instance number of the VVC
channel	t_channel	TX, RX or ALL_CHANNELS	The VVC channel of the VVC instance

## VVC functional parameters

Name	Type	Example(s)	Description
data	std_logic_vector	x"FF"	The data to be transmitted (in uart_transmit) or the expected data (in uart_expect).
msg	string	"Send to DUT"	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.
max_receptions	natural	1	The maximum number of receptions before the expected data must be found. Exceeding this limit results in an alert 'alert_level'.
timeout	time	100 ns	The maximum time to pass before the expected data must be found. Exceeding this limit results in an alert 'alert_level'.
scope	string	"UART VVC"	A string describing the scope from which the log/alert originates.

## VVC entity signals

Name	Type	Direction	Description
clk	std_logic	Input	VVC Clock signal
uart_vvc_rx	std_logic	Input	UART VVC RX signal
uart_vvc_tx	std_logic	Inout	UART VVC TX signal

## VVC entity generic constants

Name	Type	Default	Description
GC_DATA_WIDTH	natural	8	Bits in the UART byte
GC_INSTANCE_IDX	natural	1	Instance number to assign the VVC
GC_CHANNEL	t_channel	TX/RX	Channel to be assigned to this leaf VVC (only used in TX or RX implementations, not in the uart_vvc.vhd wrapper).
GC_UART_CONFIG	t_uart_bfm_config	C_UART_BFM_CONFIG_DEFAULT	Configuration for the UART BFM, see UART 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
<b>uart_transmit()</b>	<p><b>uart_transmit (VVCT, vvc_instance_idx, channel, data   {num_words, randomisation}, msg, [scope])</b></p> <p>The <code>uart_transmit()</code> VVC procedure adds a transmit command to the UART TX VVC executor queue, that will run as soon as all preceding commands have completed. The <code>uart_transmit()</code> command has two variants using either just data for a basic single transaction, or <code>num_words</code> + <code>randomisation</code> for a more advanced version. When the basic transmit command is scheduled to run, the executor calls the UART BFM <code>uart_transmit()</code> procedure, described in the UART BFM QuickRef. The <code>uart_transmit()</code> procedure can only be called using the UART TX channel, i.e. setting 'channel' to 'TX'.</p> <p>When the more advanced <code>randomisation</code> command is applied the basic BFM <code>uart_transmit()</code> transaction is executed <code>num_words</code> times with new random data each time – according to the given <code>randomisation</code> profile.</p> <p>Current defined <code>randomisation</code> profiles are: <code>RANDOM</code>: Standard uniform random. This is provided as an example.</p> <p>Errors may be injected – depending on the <code>error_injection_config</code> sub-record within the <code>vvc_config</code></p> <p>Example:</p> <pre>uart_transmit(UART_VVCT, 1, TX, x"0D", "Transmitting carriage return to DUT UART instance 1", C_SCOPE); DRAFT uart_transmit(UART_VVCT, 1, TX, 5, RANDOM, "Sending 5 random bytes to DUT UART instance 1");</pre>
<b>uart_receive()</b>	<p><b>uart_receive (VVCT, vvc_instance_idx, channel, [TO_SB], msg, [alert_level, [scope]])</b></p> <p>The <code>uart_receive()</code> VVC procedure adds a receive command to the UART RX VVC executor queue, that will run as soon as all preceding commands have completed. When the receive command is scheduled to run, the executor calls the UART BFM <code>uart_receive()</code> procedure, described in the UART BFM QuickRef. The received data from DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the received data will be stored in the VVC for a potential future fetch (see example with <code>fetch_result</code> below). The <code>uart_receive()</code> procedure can only be called using the UART RX channel, i.e. setting 'channel' to 'RX'.</p> <p>If the option <code>TO_SB</code> is applied the received data will be sent to the UART_VVC dedicated scoreboard where it will be checked against the expected value (provided by the testbench).</p> <p>Example:</p> <pre>uart_receive (UART_VVCT, 1, RX, "Receiving from DUT UART instance 1", ERROR, C_SCOPE); uart_receive (UART_VVCT, 1, RX, TO_SB, "Receiving data from DUT UART instance 1 and passing on to Scoreboard", ERROR, C_SCOPE);</pre> <p><b>Example with <code>fetch_result()</code> call:</b> Result is placed in <code>v_data</code></p> <pre>variable v_cmd_idx      : natural;                                -- Command index for the last read variable v_data         : bitvis_vip_uart.vvc_cmd_pkg.t_vvc_result; -- Result from read (...) uart_receive(UART_VVCT, 1, RX, "Receiving from DUT UART instance 1"); v_cmd_idx := get_last_received_cmd_idx(UART_VVCT, 1); await_completion(UART_VVCT,1, v_cmd_idx, 1 us, "Wait for receive to finish"); fetch_result(UART_VVCT,1, v_cmd_idx, v_data, "Fetching result from receive operation");</pre>

## uart\_expect()

**uart\_expect(VVCT, vvc\_instance\_idx, channel, data, msg, [max\_receptions, [timeout, [alert\_level, [scope]]]])**

The `uart_expect()` VVC procedure adds an expect command to the UART VVC executor queue, which will run as soon as all preceding commands have completed. When the expect command is scheduled to run, the executor calls the UART BFM `uart_expect()` procedure, described in the UART BFM QuickRef. The received data will not be stored by this procedure. The `uart_expect()` procedure can only be called using the UART RX channel, i.e. setting 'channel' to 'RX'.

Examples:

```
uart_expect(UART_VVCT, 1, RX, x"0D", "Expecting carriage return from DUT UART instance 1");
uart_expect(UART_VVCT, 1, RX, C_CR_BYTE, "Expecting carriage return from DUT UART instance 1", 5, 10 ms, ERROR, C_SCOPE);
```

## 2 VVC Configuration

Record element	Type	C_UART_VVC_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_UART_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 triggered if command count 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_uart_bfm_config	C_UART_BFM_CONFIG_DEFAULT	Configuration for UART BFM. See QuickRef for UART BFM
error_injection	t_vvc_error_injection	C_ERROR_INJECTION_INACTIVE	Sets up the error injection policy. Will use this to set the error injection record inside the bfm_config. See table below.
bit_rate_checker	t_bit_rate_checker	C_BIT_RATE_CHECKER_DEFAULT	Configure the UART property checker behaviour.
msg_id_panel	t_msg_id_panel	C_VVC_MSG_ID_PANEL_DEFAULT	VVC dedicated message ID panel. See section 16 of <a href="#">uvm_vvc_framework/doc/UVVM_VVC_Framework_Essential_Mechanisms.pdf</a> 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_uart_vvc_config(TX,1).inter_bfm_delay.delay_in_time := 10 ms;
shared_uart_vvc_config(RX,1).bfm_config.num_data_bits      := 8;
```

## VVC Error injection record (inside the VVC configuration record above)

Record element	Type	DEFAULT	Description
parity_bit_error_prob	real	-1,0	The probability that the VVC will request a parity_bit_error when calling a BFM transmission procedure. (See BFM doc)
stop_bit_error_prob	real	-1,0	The probability that the VVC will request a stop_bit_error when calling a BFM transmission procedure. (See BFM doc)

Note 1: A value of 1.0 means every transmission should have this error injection, whereas 0.0 means error injection is turned off. Anything in between means randomisation with the given probability

Note 2: The error\_injection\_config in the VVC config will override any error injection specified in the BFM config, unless set to **-1.0** (default) in which case the BFM config error injection setting will be used.

Error injection in general is explained in ‘UVVM Essential Mechanisms’ located in `uvvm_vvc_framework/doc`.

## VVC Property checking record (inside the VVC configuration record above)

Record element	Type	DEFAULT	Description
enable	boolean	FALSE	Enables or disables the complete bit rate checker
min_period	time	0,0	The minimum allowed bit period for any bit (any bit level change to the next)
alert_level	t_alert_level	ERROR	Alert generated if minimum requirement is violated

Property checking and controlling this is explained in general in ‘UVVM Essential Mechanisms’ located in `uvvm_vvc_framework/doc`.

## 3 VVC Status

The current status of the VVC can be retrieved during simulation. This is done by reading from the shared variable `shared_uart_vvc_status` record from the test sequencer. The record contains status for both channels, specified with the channel axis of the `shared_uart_vvc_status` array. 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 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 4.1 UART transaction info record fields. Transaction type: `t_base_transaction (BT)` - accessible via **`shared_uart_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.
data	slv (299 downto 0)	0x0	The data to be transmitted (in <code>uart_transmit</code> ) or the expected data (in <code>uart_expect</code> ).
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.

## 5 Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the `T0_SB` parameter in supported method calls, i.e. `uart_receive ()`. Note that the data is only stored in the scoreboard and not accessible with the `fetch_result()` method when the `T0_SB` parameter is applied.

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 SBI VVC scoreboard is accessible from the testbench as a shared variable `UART_VVC_SB`, located in the `vvm_methods_pkg.vhd`. All of the listed Generic Scoreboard commands are available for the UART VVC scoreboard using this shared variable.

## 6 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`.

Note that each channel is counted in the number of registered VVCs in the VVC activity register, thus the UART VVC is counted as two VVCs. More information can be found in UVVM Essential Mechanisms PDF in the UVVM VVC Framework doc folder.

## 7 Additional Documentation

Additional documentation about UVVM and its features can be found under `"/uvvm_vvc_framework/doc/`". For additional documentation on the UART protocol, please see the UART specification.

## 8 Compilation

The UART 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**
- **UART BFM**
- **Bitvis VIP Scoreboard**

Before compiling the UART VVC, make sure 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 UART VVC:

Compile to library	File	Comment
bitvis_vip_uart	uart_bfm_pkg.vhd	UART BFM
bitvis_vip_uart	transaction_pkg.vhd	UART transaction package with DTT types, constants etc.
bitvis_vip_uart	vvc_cmd_pkg.vhd	UART VVC command types and operations
bitvis_vip_uart	[ monitor_cmd_pkg.vhd ]	UART Monitor package. Only include this file if you intend to use Monitor.
bitvis_vip_uart	../uvvm_vvc_framework/src_target_dependent/td_target_support_pkg.vhd	UVVM VVC target support package, compiled into the UART VVC library.
bitvis_vip_uart	../uvvm_vvc_framework/src_target_dependent/td_vvc_framework_common_methods_pkg.vhd	UVVM framework common methods compiled into the UART VVC library
bitvis_vip_uart	vvc_methods_pkg.vhd	UART VVC methods
bitvis_vip_uart	../uvvm_vvc_framework/src_target_dependent/td_queue_pkg.vhd	UVVM queue package for the VVC
bitvis_vip_uart	../uvvm_vvc_framework/src_target_dependent/td_vvc_entity_support_pkg.vhd	UVVM VVC entity methods compiled into the UART VVC library
bitvis_vip_uart	uart_rx_vvc.vhd	UART RX VVC
bitvis_vip_uart	uart_tx_vvc.vhd	UART TX VVC
bitvis_vip_uart	uart_vvc.vhd	UART VVC wrapper for the RX and TX VVCs
bitvis_vip_uart	[ uart_monitor.vhd ]	UART Monitor. Only include this file if you intend to use Monitor.
bitvis_vip_uart	vvc_context.vhd	Common UART VIP use declarations.

## 9 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 UART TX and RX.

The given VIP complies with the basic UART protocol and thus allows a normal access towards a UART interface. This VIP is not a UART 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.