**UART VVC** –Quick Reference

**VVC**

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 Peripheral 1”);  uart\_transmit(UART\_VVCT, 1, TX, 5, RANDOM, “Sending 5 random bytes”); |

*uart\_vvc.vhd*

*uart\_rx\_vvc.vhd*

*uart\_tx\_vvc.vhd*

|  |
| --- |
| uart\_receive (VVCT, vvc\_instance\_idx, channel, [TO\_SB,] msg, [alert\_level, [scope]]) |
| Example: uart\_receive(UART\_VVCT, 1, RX, “Receive from Peripheral 1”);  uart\_receive (UART\_VVCT, 1, RX, TO\_SB, “Receiving data 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 Peripheral 1”); |

UART VVC Configuration record **´vvc\_config´ --** accessible via **shared\_uart\_vvc\_config**

**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()**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **C\_UART\_VVC\_CONFIG\_DEFAULT** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_UART\_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\_uart\_bfm\_config | C\_UART\_BFM\_CONFIG\_DEFAULT |
| error\_injection | t\_vvc\_error\_injection | C\_ERROR\_INJECTION\_INACTIVE |
| bit\_rate\_checker | t\_bit\_rate\_checker | C\_BIT\_RATE\_CHECKER\_DEFAULT |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT |
|  |  |  |

UART VVC Status record signal **´vvc\_status´ --** accessible via **shared\_uart\_vvc\_status**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** |  |
| current\_cmd\_idx | natural |  | |
| previous\_cmd\_idx | natural |  |
| pending\_cmd\_cnt | natural |  |



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 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. |
| 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 [ ].*

# VVC procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **uart\_transmit()** | **uart\_transmit (VVCT, vvc\_instance\_idx, channel, data | {num\_words, randomisation}, msg, [scope])**  The uart\_transmit() 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 uart\_transmit() command has two variants using either just data for a basic single transaction, or num\_words + randomisation for a more advanced version.  When the basic transmit command is scheduled to run, the executor calls the UART BFM uart\_transmit() procedure, described in the UART BFM QuickRef. The uart\_transmit() procedure can only be called using the UART TX channel, i.e. setting ‘channel’ to ‘TX’.  When the more advanced randomisation command is applied the basic BFM uart\_transmit() transaction is executed num\_words times with new random data each time – according to the given randomisation profile.  Current defined randomisation profiles are: RANDOM: Standard uniform random. This is provided as an example.  Errors may be injected – depending on the error\_injection\_config sub-record within the vvc\_config  Example:  uart\_transmit(UART\_VVCT, 1, TX, x”0D”, “Transmitting carriage return to Peripheral 1”, C\_SCOPE);  DRAFT uart\_transmit(UART\_VVCT, 1, TX, 5, RANDOM, “Sending 5 random bytes”); |
| **uart\_receive()** | **uart\_receive (VVCT, vvc\_instance\_idx, channel, [TO\_SB,] msg, [alert\_level, [scope]])**  The uart\_receive() 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 uart\_receive() 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 *fetch\_result* below). The uart\_receive() procedure can only be called using the UART RX channel, i.e. setting ‘channel’ to ‘RX’.  If the option TO\_SB 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).  Example:  uart\_receive (UART\_VVCT, 1, RX, “Receiving from Peripheral 1”, ERROR, C\_SCOPE);  uart\_receive (UART\_VVCT, 1, RX, TO\_SB, “Receiving data and passing on to Scoreboard”, ERROR, 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 : bitvis\_vip\_uart.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from read  (…)  uart\_receive(UART\_VVCT, 1, RX, “Receiving from Peripheral 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"); |
| **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 Peripheral 1”);  uart\_expect(UART\_VVCT, 1, RX, C\_CR\_BYTE, “Expecting carriage return from Peripheral 1”, 5, 10 ms, ERROR, C\_SCOPE); |

# 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 |

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 | 0,0 | 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 | 0,0 | 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. | | | | |

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. |  |  |  |

# 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 |

# 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).

|  |  |  |  |
| --- | --- | --- | --- |
| **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 uart\_transmit) or the expected data (in uart\_expect). |
| 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.

# 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. uart\_receive (). 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.

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 scoreboard is accessible from the testbench as a shared variable UART\_SB, located in the vvc\_methods\_pkg.vhd. All of the listed Generic Scoreboard commands are available for the UART VVC using this shared variable.

# 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.

Note that each channel is counted in the number of registered VVCs in the activity watchdog 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.

# 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.

# Compilation

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

* ***UVVM Utility Library (UVVM-Util), version 2.13.0 and up***
* ***UVVM VVC Framework, version 2.8.0 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. |

# 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

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.

**INTELLECTUAL**

**PROPERTY**