**Common VVC Methods (Command Distribution Methods)** –Quick Reference

|  |
| --- |
| [**await\_completion**](#await_completion) (vvc\_target, vvc\_instance\_idx, [vvc\_channel,] [wanted\_idx,] [timeout, [msg, [scope]]])  [**await\_completion**](#await_completion) (ANY\_OF, vvc\_list, timeout, [list\_action, [msg, [scope]]]) |
| Example: await\_completion(SBI\_VVCT, 1, 100 ns, “Waiting for all SBI commands to complete”);  Example: await\_completion(UART\_VVCT, 1, RX, v\_idx, 100 ns, “Waiting for UART receive to complete”, C\_SCOPE);  Example: await\_completion(ANY\_OF, my\_vvc\_list, 1 ms, KEEP\_LIST, "Waiting for any VVC in the list to complete", C\_SCOPE); |

|  |
| --- |
| [**get\_last\_received\_cmd\_idx**](#get_last_received_cmd_idx) (vvc\_target, vvc\_instance\_idx, [vvc\_channel, [scope]]) |
| Example: v\_cmd\_idx := get\_last\_received\_cmd\_idx (SBI\_VVCT, 1);  Example: v\_cmd\_idx := get\_last\_received\_cmd\_idx (UART\_VVCT, 1, RX); |

|  |
| --- |
| [**terminate\_current\_command**](#terminate_currant_command) (vvc\_target, vvc\_instance\_idx, [vvc\_channel, [msg, [scope]]]) |
| Example: terminate\_current\_command(SBI\_VVCT, 1); |

|  |
| --- |
| [**terminate\_all\_commands**](#terminate_all_commands) (vvc\_target, vvc\_instance\_idx, [vvc\_channel, [msg, [scope]]]) |
| Example: terminate\_all\_commands(UART\_VVCT, 1, RX); |

|  |
| --- |
| [**await\_any\_completion**](#await_completion) () |
| *Note: this procedure will be deprecated in future releases, see page 4 for syntax and more info.* |

|  |
| --- |
| [**enable\_log\_msg**](#enable_log_msg) (vvc\_target, vvc\_instance\_idx, [vvc\_channel,] msg\_id, [msg, [quietness, [scope]]]) |
| Example: enable\_log\_msg(UART\_VVCT, 1, RX, ID\_BFM); |

|  |
| --- |
| [**disable\_log\_msg**](#disable_log_msg) (vvc\_target, vvc\_instance\_idx, [vvc\_channel,] msg\_id, [msg, [quietness, [scope]]]) |
| Example: disable\_log\_msg(SBI\_VVCT, 1, ID\_BFM); |

|  |
| --- |
| [**flush\_command\_queue**](#flush_command_queue) (vvc\_target, vvc\_instance\_idx, [vvc\_channel,] [msg, [scope]]) |
| Example: flush\_command\_queue(AXILITE\_VVCT, 1); |

|  |
| --- |
| [**fetch\_result**](#fetch_result) (vvc\_target, vvc\_instance\_idx, [vvc\_channel,] wanted\_idx, result, [fetch\_is\_accepted,] [msg, [alert\_level, [scope]]]) |
| Example: fetch\_result(SBI\_VVCT, 1, v\_idx, v\_result, v\_fetch\_is\_accepted); |

|  |
| --- |
| [**insert\_delay**](#insert_delay) (vvc\_target, vvc\_instance\_idx, [vvc\_channel,] delay, [msg, [scope]]) |
| Example: insert\_delay(SBI\_VVCT, 1,100 ns);  Example: insert\_delay(UART\_VVCT, 1, TX, 10); -- 10 Clock cycles delay using the VVC clk |



UVVM methods - target parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| vvc\_target | 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 used in this method |
| vvc\_channel | t\_channel | TX, RX or ALL\_CHANNELS | The VVC channel of the VVC instance used in this method |
| vvc\_list | t\_vvc\_list | v\_vvc\_list | A list of protected type containing one or several VVC IDs (name, instance, channel, command index).  VVC IDs can be added to the list by using the procedure **add(name, instance, [channel,] [cmd\_idx])**. The name is a string that should match the C\_VVC\_NAME in the VVC’s vvc\_methods\_pkg.vhd. |
| void | t\_void | VOID | An empty input parameter for procedure waiting for UVVM to be initialized. |

UVVM methods - functional parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| wanted\_idx | natural | 50 | The index to be fetched or awaited |
| list\_action | t\_list\_action | KEEP\_LIST, CLEAR\_LIST | An enumerated type to either keep the VVC IDs or remove them from the list after await\_completion() has finished. |
| timeout | time | 100 ns | The maximum time to await completion of a specified command, or all pending commands. An alert of severity ERROR will be triggered if the awaited time is equal to the specified timeout. |
| msg | string | “Awaiting CR from UART” | A message parameter to be appended to the log when the method is executed. |
| msg\_id | t\_msg\_id | ID\_SEQUENCER | The ID to enable/disable with enable/disable\_log\_msg(). For more info, see the UVVM-Util documentation. |
| result | t\_vvc\_result | v\_result | The output where the fetched data is to be placed with fetch\_result() |
| fetch\_is\_accepted | boolean | v\_fetch\_is\_accepted | Output containing a Boolean that states if the fetch command was accepted or not. Will be false if the specified command index has not been stored. |
| alert\_level | t\_alert\_level | TB\_WARNING | The alert level used for the alert which occurs when a fetch\_result() command is not accepted |
| delay | time or natural | 100 ns or 10 | Delay to be inserted in the insert\_delay() procedure, either as time or number of clock cycles |
| quietness | t\_quietness | QUIET | The logging of the command can be turned off by setting quietness=QUIET. |
| scope | string | “Sequencer 1” | A string describing the scope from which the log/alert originates. |

UVVM VVC Framework command broadcasting and multicasting

Commands in UVVM can be distributed to all instances of a VVC or to all VVCs using dedicated parameters.

|  |  |
| --- | --- |
| **Command Parameter** | **Description** |
| **VVC\_BROADCAST** | The VVC\_BROADCAST command parameter can be used when a command is to target all VVCs within the test environment, reducing the number of command instructions needed in the testbench.  Example**:**  enable\_log\_msg(VVC\_BROADCAST, ALL\_MESSAGES); -- enable logging for all VVCs  await\_completion(VVC\_BROADCAST, 10 us); -- wait for all VVCs to complete |
| **ALL\_INSTANCES** | The ALL\_INSTANCES command parameter can be used when a command is targeting all instances of a VVC within the test environment, reducing the number of command instructions needed in the testbench.  Example**:**  enable\_log\_msg(SBI\_VVCT, ALL\_INSTANCES, ALL\_MESSAGES); -- enable logging for all instances of SBI\_VVCT  await\_completion(SBI\_VVCT, ALL\_INSTANCES, 100 ns); -- wait for all instances of SBI\_VVCT to complete |
| **ALL\_CHANNELS** | The ALL\_CHANNELS command parameter can be used when a command is targeting all channels of a VVC within the test environment, reducing the number of command instructions needed in the testbench.  Example**:**  enable\_log\_msg(UART\_VVCT, 1, ALL\_CHANNELS, ALL\_MESSAGES); -- enable logging for all channels of SBI\_VVCT instance 1  await\_completion(UART\_VVCT, ALL\_INSTANCES, ALL\_CHANNELS, 100 ns); -- wait for all instances and channels of UART\_VVCT to complete |
| **C\_VVCT\_ALL\_INSTANCES** | See description above. C\_VVCT\_ALL\_INSTANCES = ALL\_INSTANCES.  Warning! This command parameter might be removed in a future release and we encourage the use of ALL\_INSTANCES. |

UVVM VVC Framework Common Methods details

All VVC procedures are defined in the UVVM VVC framework common methods package, td\_vvc\_framework\_common\_methods\_pkg.vhd

# UVVM VVC Framework Common Methods details and examples

|  |  |
| --- | --- |
| **Method** | **Description** |
| **await\_completion()** | **await\_completion(vvc\_target, vvc\_instance\_idx, timeout, msg, scope)**  **await\_completion(vvc\_target, vvc\_instance\_idx, wanted\_idx, timeout, msg, scope)**  **await\_completion(vvc\_target, vvc\_instance\_idx, vvc\_channel, timeout, msg, scope)**  **await\_completion(vvc\_target, vvc\_instance\_idx, vvc\_channel, wanted\_idx, timeout, msg, scope)**  **await\_completion(ANY\_OF, vvc\_list, timeout, list\_action, msg, scope)**  Tells the VVC to await the completion of either all pending commands or a specified command index.  A message will be logged before and at the end of the wait.  The procedure will report an alert if not all commands have completed within the specified time, *timeout*. The severity of this alert will be TB\_ERROR.  It is also possible multicast to ALL\_INSTANCES or ALL\_CHANNELS of a VVC.  To await the completion of one out of several VVCs in a group use the overload with the vvc\_list.  The vvc\_list of type t\_vvc\_list (protected type) is a local variable that needs to be declared in the sequencer. The list\_action default is to clear the list.  This overload will block the sequencer while waiting, but not the VVCs, so they can continue to receive commands from other sequencers.  **Important**: to use the vvc\_list, the package uvvm\_vvc\_framework.ti\_protected\_types\_pkg.all must be included in the testbench. Note that the command with the vvc\_list requires VVCs supporting the VVC activity register introduced in UVVM release v2020.05.19  Examples:  await\_completion(SBI\_VVCT, 1, 16 ns, "Wait for SBI instance 1 to finish", C\_SCOPE);  await\_completion(SBI\_VVCT, 1, v\_cmd\_idx, 100 ns, "Wait for sbi\_read to finish", C\_SCOPE);  Multicast:  await\_completion(SBI\_VVCT, ALL\_INSTANCES, 100 ns, "Wait for all SBI instances to finish", C\_SCOPE);  await\_completion(UART\_VVCT, 1, ALL\_CHANNELS, 100 ns, "Wait for all UART channels from instance 1 to finish", C\_SCOPE);  Using vvc\_list:  variable my\_vvc\_list : t\_vvc\_list;  my\_vvc\_list.add(“SBI\_VVC”, 1);  my\_vvc\_list.add(“AXISTREAM\_VVC”, 3, v\_cmd\_idx);  my\_vvc\_list.add(“UART\_VVC”, ALL\_INSTANCES, ALL\_CHANNELS);  await\_completion(ANY\_OF, my\_vvc\_list, 1 ms, KEEP\_LIST, "Wait for any VVC in the list to finish", C\_SCOPE); |
| **await\_any\_completion()** | **Replaced by await\_completion(ANY\_OF, vvc\_list, timeout, list\_action, msg, scope) above to allow VVCs to accept commands while waiting for completion.**  **This command still works as previously, but with less functionality than the new await\_completion(ANY\_OF, …)**  **Warning! This procedure will soon be deprecated and removed.**  **For details and examples for using this call see UVVM release v2020.05.12 or any earlier releases.**  **await\_any\_completion(vvc\_target, vvc\_instance\_idx, [vvc\_channel,] [wanted\_idx,] lastness, [timeout, [msg, [await\_completion\_idx, [scope]]]])** |
| **disable\_log\_msg()** | **disable\_log\_msg(vvc\_target, vvc\_instance\_idx, msg\_id, msg, quietness, scope)**  **disable\_log\_msg(vvc\_target, vvc\_instance\_idx, vvc\_channel, msg\_id, msg, quietness, scope)**  Instruct the VVC to disable a given log ID. This call will be forwarded to the UVVM Utility Library disable\_log\_msg function. For more information about the disable\_log\_msg() method, please refer to the UVVM-Util QuickRef.  It is also available as a broadcast to all VVCs.  Examples:  disable\_log\_msg(SBI\_VVCT, 1, ID\_LOG\_BFM, “Disabling SBI BFM logging”);  disable\_log\_msg(UART\_VVCT, 1, TX, ID\_LOG\_BFM, “Disabling UART TX BFM logging”, NON\_QUIET, C\_SCOPE);  Broadcast:  disable\_log\_msg(VVC\_BROADCAST, ALL\_MESSAGES, "Disables all messages in all VVCs", NON\_QUIET, C\_SCOPE); |
| **enable\_log\_msg()** | **enable\_log\_msg(vvc\_target, vvc\_instance\_idx, msg\_id, msg, quietness, scope)**  **enable\_log\_msg(vvc\_target, vvc\_instance\_idx, vvc\_channel, msg\_id, msg, quietness, scope)**  Instruct the VVC to enable a given log ID. This call will be forwarded to the UVVM Utility Library enable\_log\_msg function. For more information about the enable\_log\_msg() method, please refer to the UVVM-Util QuickRef.  It is also available as a broadcast to all VVCs.  Examples:  enable\_log\_msg(SBI\_VVCT, 1, ID\_LOG\_BFM, “Enabling SBI BFM logging”);  enable\_log\_msg(UART\_VVCT, 1, TX, ID\_LOG\_BFM, “Enabling UART TX BFM logging”, NON\_QUIET, C\_SCOPE);  Broadcast:  enable\_log\_msg(VVC\_BROADCAST, ID\_LOG\_BFM, "Enabling BFM logging for all VVCs", NON\_QUIET, C\_SCOPE); |
| **flush\_command\_queue()** | **flush\_command\_queue(vvc\_target, vvc\_instance\_idx, msg, scope)**  **flush\_command\_queue(vvc\_target, vvc\_instance\_idx, vvc\_channel, msg, scope)**  Flushes the VVC command queue for the specified VVC target/channel. The procedure will log information with log ID ID\_IMMEDIATE\_CMD.  It is also available as a broadcast to all VVCs.  Example:  flush\_command\_queue(SBI\_VVCT, 1, “Flushing command queue”, C\_SCOPE);  Broadcast:  flush\_command\_queue(VVC\_BROADCAST, "Flushing command queues", C\_SCOPE); |
| **fetch\_result()** | **fetch\_result(vvc\_target, vvc\_instance\_idx, wanted\_id, result, msg, alert\_level, scope)**  **fetch\_result(vvc\_target, vvc\_instance\_idx, vvc\_channel, wanted\_id, result, msg, alert\_level, scope)**  **fetch\_result(vvc\_target, vvc\_instance\_idx, wanted\_id, result, fetch\_is\_accepted, msg, alert\_level, scope)**  **fetch\_result(vvc\_target, vvc\_instance\_idx, vvc\_channel, wanted\_id, result, fetch\_is\_accepted, msg, alert\_level, scope)**  Fetches a stored result using the command index. A result is stored when using e.g. the read or receive commands in a VVC. The fetched result is available on the ‘result’ output. The Boolean output ‘fetch\_is\_accepted’ is used to indicate if the fetch was successful or not. A fetch can fail if e.g. the wanted\_id did not have a result to store, or the wanted\_id read has not yet been executed. Omitting the ‘fetch\_is\_accepted’ parameter causes the parameters to be checked automatically in the procedure. On successful fetch, a message with log ID ID\_UVVM\_CMD\_RESULT is logged.  Example:  fetch\_result(SBI\_VVCT,1, v\_cmd\_idx, v\_data, v\_is\_ok, "Fetching read-result", C\_SCOPE);  Full example:  sbi\_read(SBI\_VVCT, 1, C\_ADDR\_FIFO\_GET, "Read from FIFO");  v\_cmd\_idx := get\_last\_received\_cmd\_idx(SBI\_VVCT,1); -- Retrieve the command index  await\_completion(SBI\_VVCT, 1, v\_cmd\_idx, 100 ns, "Wait for sbi\_read to finish");  fetch\_result(SBI\_VVCT, 1, v\_cmd\_idx, v\_data, v\_is\_ok, "Fetching read-result");  check\_value(v\_is\_ok, ERROR, "Readback OK via fetch\_result()"); |
| **insert\_delay()** | **insert\_delay(vvc\_target, vvc\_instance\_idx, delay, msg, scope)**  **insert\_delay(vvc\_target, vvc\_instance\_idx, vvc\_channel, delay, msg, scope)**  This method inserts a delay of ‘delay’ clock cycles or ‘delay’ seconds in the VVC.  It is also available as a broadcast to all VVCs.  Examples:  insert\_delay(SBI\_VVCT,1, 100, "100T delay", C\_SCOPE);  insert\_delay(SBI\_VVCT,1, 50 ns, "50 ns delay", C\_SCOPE);  Broadcast:  insert\_delay(VVC\_BROADCAST, 50 ns, "Insert 50 ns delay to all VVCs", C\_SCOPE); |
| **terminate\_current\_command()** | **terminate\_current\_command(vvc\_target, vvc\_instance\_idx, msg, scope)**  **terminate\_current\_command(vvc\_target, vvc\_instance\_idx, vvc\_channel, msg, scope)**  This method terminates the current command in the VVC, if the currently running BFM command supports the terminate signal.  It is also available as a broadcast to all VVCs.  Example:  terminate\_current\_command(SBI\_VVCT, 1, “Terminating current command”, C\_SCOPE);  Broadcast:  terminate\_current\_command(VVC\_BROADCAST, “Terminating current command in all VVCs”, C\_SCOPE); |
| **terminate\_all\_commands()** | **terminate\_all\_commands(vvc\_target, vvc\_instance\_idx, msg, scope)**  **terminate\_all\_commands(vvc\_target, vvc\_instance\_idx, vvc\_channel, msg, scope)**  This method terminates the current command in the VVC, if the currently running BFM command supports the terminate signal. The terminate\_all\_commands() procedure also flushes the VVC command queue, removing all pending commands.  It is also available as a broadcast to all VVCs.  Example:  terminate\_all\_commands(SBI\_VVCT, 1, “Terminating all commands”, C\_SCOPE);  Broadcast:  terminate\_all\_commands(VVC\_BROADCAST,”Terminating all commands in all VVCs”, C\_SCOPE); |
| **get\_last\_received\_cmd\_idx()** | **get\_last\_received\_cmd\_idx(vvc\_target, vvc\_instance\_idx, scope)**  **get\_last\_received\_cmd\_idx(vvc\_target, vvc\_instance\_idx, vvc\_channel, scope)**  This method is used to get the command index of the last command received by the VVC interpreter. Necessary for getting the command index of a read for fetch\_result.  Example:  v\_cmd\_idx := get\_last\_received\_cmd\_idx(SBI\_VVCT, 1, C\_SCOPE); |

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**