

I2C VVC – Quick Reference

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

I2C Master

i2c_master_transmit (VVCT, vvc_instance_idx, addr, data, msg, [action_when_transfer_is_done, [scope]])

Example: `i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_ADDR, x"AF", "Sending data from master VVC to slave DUT");`

i2c_master_check (VVCT, vvc_instance_idx, addr, data, msg, [action_when_transfer_is_done, [alert_level, [scope]]])

Example: `i2c_master_check(I2C_VVCT, 1, C_SLAVE_0_ADDR, x"42", "Expect data from slave DUT");`

i2c_master_receive (VVCT, vvc_instance_idx, addr, num_bytes, [TO_SB,] msg, [action_when_transfer_is_done, [scope]])

Example: `i2c_master_receive(I2C_VVCT, 1, C_SLAVE_0_ADDR, 1, "Receive1 byte from slave DUT and store data in VVC. To be retrieved using fetch_result() ");`
`i2c_master_receive(I2C_VVCT, 1, C_SLAVE_0_ADDR, 3, TO_SB "Receive3 bytes from slave DUT and send to scoreboard for checking");`

I2C Slave

i2c_master_quick_command (VVCT, vvc_instance_idx, addr, msg, [rw_bit, [exp_ack, [action_when_transfer_is_done, [alert_level, [scope]]]])

Example: `i2c_master_quick_command(I2C_VVCT, 1, C_SLAVE_0_ADDR, "Quick Command to Slave 0");`

i2c_slave_transmit (VVCT, vvc_instance_idx, data, msg, [scope])

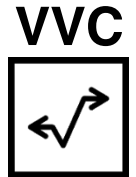
Example: `i2c_slave_transmit(I2C_VVCT, 2, x"DB", "Sending data from slave VVC to master DUT");`

i2c_slave_check (VVCT, vvc_instance_idx, {data, rw_bit}, msg, [alert_level, [scope]])

Example: `i2c_slave_check(I2C_VVCT, 2, x"42", "Expect data from master DUT");`

i2c_slave_receive (VVCT, vvc_instance_idx, num_bytes, [TO_SB,] msg, [scope])

Example: `i2c_slave_receive(I2C_VVCT, 1, 1, "Receive1 byte from slave DUT and store data in VVC. To be retrieved using fetch_result()");`
`i2c_slave_receive(I2C_VVCT, 1, 6, TO_SB, "Receive 6 bytes from slave DUT and send to scoreboard for checking");`



I2C VVC Configuration record `'vvc_config'` -- accessible via `shared_i2c_vvc_config`

| Record element | Type | C_I2C_VVC_CONFIG_DEFAULT |
|---|-------------------|---|
| inter_bfm_delay | t_inter_bfm_delay | C_I2C_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_i2c_bfm_config | C_I2C_BFM_CONFIG_DEFAULT |
| msg_id_panel | t_msg_id_panel | C_VVC_MSG_ID_PANEL_DEFAULT |
| unwanted_activity_severity | t_alert_level | C_UNWANTED_ACTIVITY_SEVERITY |

I2C VVC Status record signal `'vvc_status'` -- accessible via `shared_i2c_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()`

`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 | I2C_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"AF" | Slave address to interact with when VVC is in master mode. |
| data | std_logic_vector(7 downto 0) t_byte_array | x"94" or [x"FF", x"AA", x"DB"] | The data to be transmitted (in i2c_<master/slave>_transmit) or the expected data (in i2c_<master/slave>_check). Either a single byte or a byte array. |
| msg | string | "Send to peripheral 1" | A custom message to be appended in the log/alert |
| action_when_transfer_is_done | t_action_when_transfer_is_done | RELEASE_LINE_AFTER_TRANSFER or HOLD_LINE_AFTER_TRANSFER | This parameter sets whether the VVC (in master mode) shall occupy the bus after the current transaction is finished. 'HOLD_LINE_AFTER_TRANSFER' means that the VVC will not generate a stop condition at the end of the current transaction. When the next transaction starts, the master VVC generates a start condition that will be interpreted by the slave(s) as a repeated start condition. |
| alert_level | t_alert_level | ERROR or TB_WARNING | Set the severity for the alert that may be asserted by the method. |
| rw_bit | std_logic | '0' or '1' | Bit set in the R/W# slot of the Quick Command |
| exp_ack | boolean | true or false | Expected ack bit during a Quick Command. Can be used to e.g. identify if a slave is present on the bus. |
| scope | string | "I2C VVC" | A string describing the scope from which the log/alert originates. In a simple single sequencer typically "I2C BFM". In a verification component typically "I2C VVC". |

VVC entity generic constants

| Name | Type | Default | Description |
|--|------------------|--------------------------|--|
| GC_INSTANCE_IDX | natural | 1 | Instance number to assign the VVC |
| GC_MASTER_MODE | boolean | true | Master mode enabled when set to 'true'. The VVC may then only use the 'i2c_master_<transmit/check>' methods. When set to 'false' the VVC will act as an I2C slave and may only use the 'i2c_slave_<transmit/check>' methods. |
| GC_I2C_CONFIG | t_i2c_bfm_config | C_I2C_BFM_CONFIG_DEFAULT | Configuration for the I2C BFM, see I2C 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 GC_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 entity signals

| Name | Type | Direction | Description |
|------|-----------|-----------|----------------|
| scl | std_logic | Inout | I2C SCL signal |
| sda | std_logic | Inout | I2C SDA signal |

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 |
|------------------------------|--|
| i2c_master_transmit() | <p>i2c_master_transmit (VVCT, vvc_instance_idx, addr, data, msg, [action_when_transfer_is_done, [scope]])</p> <p>The <code>i2c_master_transmit()</code> VVC procedure adds a master transmit command to the I2C VVC executor queue, that will run as soon as all preceding commands have completed. When the master transmit command is scheduled to run, the executor calls the I2C BFM <code>i2c_master_transmit()</code> procedure, described in the I2C BFM QuickRef. The <code>i2c_master_transmit()</code> procedure can only be called when the I2C VVC is instantiated in master mode, i.e. setting the VVC entity generic constant 'GC_MASTER_MODE' to 'true'.</p> <p>Examples:</p> <pre>i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_0_ADDR, x"0D", "Transmitting data to slave 0"); i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_1_ADDR, byte_array(0 to 3), "Transmitting byte array to slave 1 without generating stop condition at the end", HOLD_LINE_AFTER_TRANSFER, C_SCOPE);</pre> |
| i2c_master_check() | <p>i2c_master_check (VVCT, instance_idx, addr, data, msg, [action_when_transfer_is_done, [alert_level, [scope]]])</p> <p>The <code>i2c_master_check ()</code> VVC procedure adds a master check command to the I2C VVC executor queue, which will run as soon as all preceding commands have completed. When the master check command is scheduled to run, the executor calls the I2C BFM <code>i2c_master_check()</code> procedure, described in the I2C BFM QuickRef. The received data will not be stored by this procedure. The <code>i2c_master_check()</code> procedure can only be called when the I2C VVC is instantiated in master mode, i.e. setting the VVC entity generic constant 'GC_MASTER_MODE' to 'true'.</p> <p>Examples:</p> <pre>i2c_master_check(I2C_VVCT, 1, C_SLAVE_0_ADDR, byte_array(0 to 20), "Expecting byte array from Slave 0"); i2c_master_check(I2C_VVCT, 1, C_SLAVE_1_ADDR, x"AD", "Expecting data from Slave 1 without generating stop condition at the end", HOLD_LINE_AFTER_TRANSFER, WARNING, C_SCOPE);</pre> |
| i2c_master_receive() | <p>i2c_master_receive (VVCT, instance_idx, channel, [TO_SB,] msg, [scope])</p> <p>The <code>i2c_master_receive()</code> VVC procedure adds a receive command to the I2C 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 I2C BFM <code>i2c_slave_receive ()</code> procedure, described in the I2C BFM QuickRef.</p> <p>The received data 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 <i>fetch_result</i> below).</p> <p>If the option <code>TO_SB</code> is applied, the received data will be sent to the I2C dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).</p> <p>Example:</p> <pre>i2c_master_receive (I2C_VVCT, 1, C_I2C_SLAVE_ADDR, 4, "Receiving 4 bytes from I2C Slave with address C_I2C_SLAVE_ADDR", C_SCOPE);</pre> <p>Example with <i>fetch_result()</i> call: Result is placed in v_byte_array</p> <pre>variable v_cmd_idx : natural; -- Command index for the last read variable v_byte_array : bitvis_vip_i2c.vvc_cmd_pkg.t_vvc_result; (...) i2c_master_receive(I2C_VVCT, 1, C_I2C_SLAVE_ADDR, 4, "Master receives 4 bytes from Slave with address C_I2C_SLAVE_ADDR"); v_cmd_idx := get_last_received_cmd_idx(I2C_VVCT, 1); await_completion(I2C_VVCT, 1, 50 ms); fetch_result(I2C_VVCT,1, v_cmd_idx, v_byte_array, "Fetching result from receive operation");</pre> |

| | |
|-----------------------------|---|
| i2c_slave_transmit() | i2c_slave_transmit (VVCT, vvc_instance_idx, data, msg, [scope]) <p>The i2c_slave_transmit() VVC procedure adds a slave transmit command to the I2C VVC executor queue, that will run as soon as all preceding commands have completed. When the slave transmit command is scheduled to run, the executor calls the I2C BFM i2c_slave_transmit() procedure, described in the I2C BFM QuickRef. The i2c_slave_transmit() procedure can only be called when the I2C VVC is instantiated in slave mode, i.e. setting the VVC entity generic constant 'GC_MASTER_MODE' to 'false'.</p> <p>Examples:</p> <pre>i2c_slave_transmit(I2C_VVCT, 2, x"0D", "Transmitting a single byte to master", C_SCOPE); i2c_slave_transmit(I2C_VVCT, 2, byte_array(0 to 9), "Transmitting an array of bytes to master", C_SCOPE);</pre> |
| i2c_slave_check() | i2c_slave_check (VVCT, instance_idx, data, msg, [alert_level, [scope]]) i2c_slave_check (VVCT, instance_idx, rw_bit, msg, [alert_level, [scope]]) <p>The i2c_slave_check () VVC procedure adds a slave check command to the I2C VVC executor queue, which will run as soon as all preceding commands have completed. When the slave check command is scheduled to run, the executor calls the I2C BFM i2c_slave_check() procedure, described in the I2C BFM QuickRef. The received data will not be stored by this procedure. The i2c_slave_check() procedure can only be called when the I2C VVC is instantiated in slave mode, i.e. setting the VVC entity generic constant 'GC_MASTER_MODE' to 'false'.</p> <p>Examples:</p> <pre>i2c_slave_check(I2C_VVCT, 2, x"0D", "Expecting data from master"); i2c_slave_check(I2C_VVCT, 2, x"0D", "Expecting data from master", WARNING, C_SCOPE); i2c_slave_check(I2C_VVCT, 2, '0', "Expecting write type Quick Command from master", WARNING, C_SCOPE);</pre> |
| i2c_slave_receive() | i2c_slave_receive (VVCT, instance_idx, num_bytes, [TO_SB,] msg, [scope]) <p>If the option TO_SB is applied, the received data will be sent to the I2C dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).</p> <p>See description and fetch_result() example in the description for i2c_master_receive()</p> <p>Example:</p> <pre>i2c_slave_receive(I2C_VVCT, 1, 1, "One byte from master to slave", C_SCOPE);</pre> |

2 VVC Configuration

| Record element | Type | C_I2C_VVC_CONFIG_DEFAULT | Description |
|---------------------------|-------------------|-------------------------------|--|
| inter_bfm_delay | t_inter_bfm_delay | C_I2C_INTER_BFM_DELAY_DEFAULT | <p>Delay between any requested BFM accesses towards the DUT.</p> <ul style="list-style-type: none"> - 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. <p>Any insert_delay() command will add to the above minimum delays, giving for instance the ability to skew the BFM starting time.</p> |
| 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_i2c_bfm_config | C_I2C_BFM_CONFIG_DEFAULT | Configuration for I2C BFM. See QuickRef for I2C 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. |
| unwanted_activity_severity | t_alert_level | C_UNWANTED_ACTIVITY_SEVERITY | Severity of alert to be initiated if unwanted activity on the DUT outputs is detected. Unwanted activity detection is enabled (ERROR) by default. |

Note: *cmd/result queue parameters in the VVC Configuration are unused and will be removed in v3.0, use instead the entity generic constants.*

The configuration record can be accessed from the Central Testbench Sequencer through the shared variable array, e.g.:

```
shared_i2c_vvc_config(1).inter_bfm_delay.delay_in_time := 10 ms;
shared_i2c_vvc_config(1).bfm_config.i2c_bit_time      := 100 ns;
```

3 VVC Status

The current status of the VVC can be retrieved during simulation. This is done by reading from the shared variable `shared_i2c_vvc_status` record from the test sequencer. The record contains status for both channels, specified with the channel axis of the `shared_i2c_vvc_status` array. The record contents can be seen below:

| Record element | Type | Description |
|-------------------------------|---------|---|
| <code>current_cmd_idx</code> | natural | Command index currently running |
| <code>previous_cmd_idx</code> | natural | Previous command index to run |
| <code>pending_cmd_cnt</code> | natural | Pending number of commands in the command queue |

4 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.

5 Unwanted Activity Detection

This VVC supports detection of unwanted activity from the DUT. This mechanism will give an alert if the DUT generates any unexpected bus activity. It assures that no data is output from the DUT when it is not expected, i.e. I2C check/receive VVC methods are not called. Once the VVC is inactive, it starts to monitor continuously on the DUT outputs. When unwanted activity is detected, the VVC issues an alert of severity.

The unwanted activity detection can be configured from the central testbench sequencer, where the severity of alert can be changed to a different value. To disable this feature in the testbench, e.g.:

```
shared_i2c_vvc_config(C_VVC_INDEX).unwanted_activity_severity := NO_ALERT;
```

If multiple I2C slave VVCs are connected to the same master DUT, unwanted activity will give alerts on the inactive slave VVCs. In order to avoid getting any false alerts, the unwanted activity must be disabled on the inactive slave VVCs. Note that the unwanted activity detection must be enabled again after the data transfer is complete.

For I2C VVC, the unwanted activity detection feature is enabled (`unwanted_activity_severity := ERROR`) by default.

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 5.1 I2C transaction info record fields. Transaction type: `t_base_transaction (BT)` - accessible via **`shared_i2c_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. |
| addr | unsigned(9 downto 0) | 0x0 | Slave address to interact with when VVC is in master mode. |
| data | t_byte_array(0 to 63) | (others => (others => '0')) | The data to be transmitted (in <code>i2c_<master/slave>_transmit</code>) or the expected data (in <code>i2c_<master/slave>_check</code>). Either a single byte or a byte array. |
| num_bytes | natural | 0 | Number of bytes to be transmitted (in <code>i2c_<master/slave>_transmit</code>) or the expected data (in <code>i2c_<master/slave>_check</code>). |
| action_when_transfer_is_done | t_action_when_transfer_is_done | RELEASE_LINE_AFTER_TRANSFER | This parameter sets whether the VVC (in master mode) shall occupy the bus after the current transaction is finished. 'HOLD_LINE_AFTER_TRANSFER' means that the VVC will not generate a stop condition at the end of the current transaction. When the next transaction starts, the master VVC generates a start condition that will be interpreted by the slave(s) as a repeated start condition. |
| exp_ack | boolean | true | Expected ack bit during a Quick Command. Can be used to e.g. identify if a slave is present on the bus. |
| rw_bit | sl | 0 | Bit set in the R/W# slot of the Quick Command |
| 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. |

7 Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the `T0_SB` parameter in supported method calls, e.g. `i2c_master_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.

The I2C scoreboard is per default a 64 bits wide standard logic vector. When sending expected data to the scoreboard, where the data width is smaller than the default scoreboard width, we recommend zero-padding the data with the `pad_i2d_sb()` function. I.e. `I2C_VVC_SB.add_expected(<I2C VVC instance number>, pad_i2c_sb(<exp data>))`;

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 I2C scoreboard is accessible from the testbench as a shared variable `I2C_VVC_SB`, located in the `vvc_methods_pkg.vhd`. All of the listed Generic Scoreboard commands are available for the I2C VVC scoreboard using this shared variable.

8 Additional Documentation

Additional documentation about UVVM and its features can be found under `"/uvvm_vvc_framework/doc/".`

For additional documentation on the I2C protocol, please see the NXP I2C specification "UM10204 I2C-bus specification and user manual Rev. 6", available from NXP Semiconductors.

9 Compilation

The I2C VVC must be compiled with VHDL 2008.

It is dependent on the following libraries

- **UVVM Utility Library (UVVM-Util), version 2.19.5 and up**
- **UVVM VVC Framework, version 2.12.7 and up**
- **I2C BFM**
- **Bitvis VIP Scoreboard**

Before compiling the I2C 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 I2C VVC:

| Compile to library | File | Comment |
|--------------------|--|--|
| bitvis_vip_i2c | i2c_bfm_pkg.vhd | I2C BFM |
| bitvis_vip_i2c | transaction_pkg.vhd | I2C transaction package with DTT types, constants etc. |
| bitvis_vip_i2c | vvc_cmd_pkg.vhd | I2C VVC command types and operations |
| bitvis_vip_i2c | ../uvvm_vvc_framework/src_target_dependent/td_target_support_pkg.vhd | UVVM VVC target support package, compiled into the I2C VVC library |
| bitvis_vip_i2c | ../uvvm_vvc_framework/src_target_dependent/td_vvc_framework_common_methods_pkg.vhd | UVVM framework common methods compiled into the I2C VVC library |
| bitvis_vip_i2c | vvc_methods_pkg.vhd | I2C VVC methods |
| bitvis_vip_i2c | ../uvvm_vvc_framework/src_target_dependent/td_queue_pkg.vhd | UVVM queue package, compiled into the I2C VVC library |
| bitvis_vip_i2c | ../uvvm_vvc_framework/src_target_dependent/td_vvc_entity_support_pkg.vhd | UVVM VVC entity methods compiled into the I2C VVC library |
| bitvis_vip_i2c | i2c_vvc.vhd | I2C 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 I2C.

The given VIP complies with the basic I2C protocol and thus allows a normal access towards an I2C interface. This VIP is not an I2C protocol checker.

For a more advanced VIP please contact UVVM at info@uvvm.org

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.