

I2C VVC – Quick Reference

i2c_master_transmit (VVCT, vvc_instance_idx, addr, data, msg, [continue])

Example: i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_ADDR, x"AF", "Sending data from master VVC to slave DUT");
 Example: i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_ADDR, byte_array(0 to 3), "Sending data from master VVC to slave DUT");
 Example: i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_ADDR, x"AF", "Sending data from master VVC to slave DUT without generating stop condition at the end", true);

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

Example: i2c_master_check(I2C_VVCT, 1, C_SLAVE_0_ADDR, x"42", "Expect data from slave DUT");
 Example: i2c_master_check(I2C_VVCT, 1, C_SLAVE_0_ADDR, byte_array(0 to 10), "Expect data from slave DUT");
 Example: i2c_master_check(I2C_VVCT, 1, C_SLAVE_1_ADDR, x"43", "Expect data from slave DUT. Not generating stop condition at the end of data reception", true);
 Example: i2c_master_check(I2C_VVCT, 1, C_SLAVE_2_ADDR, x"44", "Expect data from slave DUT.", false, WARNING);

i2c_master_receive (VVCT, vvc_instance_idx, addr, num_bytes, msg, [continue])

Example: i2c_master_receive(I2C_VVCT, 1, C_SLAVE_0_ADDR, 1, "Receive1 byte from slave DUT");

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

Example: i2c_master_quick_command(I2C_VVCT, 1, C_SLAVE_0_ADDR, "Quick Command to Slave 0");
 Example: i2c_master_quick_command(I2C_VVCT, 1, C_SLAVE_1_ADDR, "Detailed Quick Command", '0', true, false, ERROR);

i2c_slave_transmit (VVCT, vvc_instance_idx, data, msg)

Example: i2c_slave_transmit(I2C_VVCT, 2, x"DB", "Sending data from slave VVC to master DUT");
 Example: i2c_slave_transmit(I2C_VVCT, 2, byte_array(0 to 10), "Sending data from slave VVC to master DUT");

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

Example: i2c_slave_check(I2C_VVCT, 2, x"42", "Expect data from master DUT");
 Example: i2c_slave_check(I2C_VVCT, 2, byte_array(0 to 3), "Expect data from master DUT");
 Example: i2c_slave_check(I2C_VVCT, 2, x"42", "Expect data from master DUT", WARNING);
 Example: i2c_slave_check(I2C_VVCT, 1, '1', "Expecting Read type Quick Command from Master DUT");

i2c_slave_receive (VVCT, vvc_instance_idx, num_bytes, msg)

Example: i2c_slave_receive(I2C_VVCT, 1, 1, "Receive1 byte from slave DUT");



i2c_vvc.vhd

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

Parameter name	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

I2C VVC Status record signal 'vvc_status' -- accessible via shared_i2c_vvc_status

Parameter name	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" [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
continue	boolean	false	This parameter sets whether the VVC (in master mode) shall occupy the bus after the current transaction is finished. 'True' means that the VVC shall continue to occupy the bus, which in turn 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.

VVC entity signals

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

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 command queue exceeds this count. Used for early

GC_RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY		t_alert_level	WARNING	warning if result queue is almost full. Will be ignored if set to 0.
				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.uvvm_methods_pkg` and `uvvm_vvc_framework.uvvm_support_pkg` (common VVC procedures)

1 VVC procedure details and examples

Procedure	Description
<code>i2c_master_transmit()</code>	<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><code>i2c_master_transmit (VVCT, vvc_instance_idx, addr, data, msg, [continue])</code></p> <p>e.g.:</p> <ul style="list-style-type: none"> - <code>i2c_master_transmit(I2C_VVCT, 1, C_SLAVE_0_ADDR, x"0D", "Transmitting data to slave 0");</code> - <code>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", true);</code>
<code>i2c_master_check()</code>	<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><code>i2c_master_check (VVCT, instance_idx, addr, data, msg, [continue, [alert_level]])</code></p> <p>e.g.</p> <ul style="list-style-type: none"> - <code>i2c_master_check(I2C_VVCT, 1, C_SLAVE_0_ADDR, byte_array(0 to 20), "Expecting byte array from Slave 0");</code> - <code>i2c_master_check(I2C_VVCT, 1, C_SLAVE_1_ADDR, x"AD", "Expecting data from Slave 1 without generating stop condition at the end", true, WARNING);</code>

i2c_master_receive()

The i2c_master_receive() VVC procedure adds a receive command to the SPI 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 I2C BFM i2c_slave_receive () procedure, described in the I2C BFM QuickRef.

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 *fetch_result* below).

There are two requirements for running the i2c_master_receive() procedure:

- The 'channel' parameter must be set to 'RX'
- The VVC entity with instance index corresponding to the 'instance_idx' parameter must have the generic constant GC_MASTER_MODE set to FALSE.

i2c_master_receive (VVCT, instance_idx, channel, msg)

e.g.

```
- i2c_master_receive (I2C_VVCT, 1, C_I2C_SLAVE_ADDR, 4, "Receiving 4 bytes from I2C Slave with address C_I2C_SLAVE_ADDR");
```

Example with fetch_result() call: Result is placed in **v_byte_array**

```
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 := shared_cmd_idx;
await_completion(I2C_VVCT, 1, 50 ms);
fetch_result(I2C_VVCT,1, v_cmd_idx, v_byte_array, "Fetching result from receive operation");
```

i2c_slave_transmit()

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'.

i2c_slave_transmit (VVCT, vvc_instance_idx, data, msg)

e.g.:

- i2c_slave_transmit(I2C_VVCT, 2, x"0D", "Transmitting a single byte to master");
- i2c_slave_transmit(I2C_VVCT, 2, byte_array(0 to 9), "Transmitting an array of bytes to master");

i2c_slave_check()

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'.

i2c_slave_check (VVCT, instance_idx, data, msg, [alert_level])

i2c_slave_check (VVCT, instance_idx, rw_bit, msg, [alert_level])

e.g.

- 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);
- i2c_slave_check(I2C_VVCT, 2, '0', "Expecting write type Quick Command from master", WARNING);

i2c_slave_receive()

See description and fetch_result() example in the description for i2c_master_receive()

i2c_slave_receive (VVCT, instance_idx, num_bytes, msg)

e.g.

- i2c_slave_receive(I2C_VVCT, 1, 1, "One byte from master to slave");

2 VVC Configuration

Name	Type	C_I2C_VVC_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_I2C_INTER_BFM_DELAY_DEFAULT	Specified delay between BFM accesses from the VVC. If parameter delay_type is set to NO_DELAY, BFM accesses will be back to back, i.e. no delay.
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 command 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

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:

Name	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 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.

5 Compilation

The I2C VVC must be compiled with VHDL 2008.

It is dependent on the following libraries

- UVVM Utility Library (UVVM-Util), version 1.0.0 and up
- UVVM VVC Framework, version 1.0.0 and up
- I2C BFM

Before compiling the I2C VVC, make sure that `uvvm_vvc_framework` and `uvvm_util` have been compiled.

Compile order for the I2C VVC:

Compile to library	File	Comment
bitvis_vip_i2c	i2c_bfm_pkg.vhd	I2C BFM
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

6 Simulator compatibility and setup

This VVC has been compiled and tested with Modelsim version 10.3d and Riviera-PRO version 2015.10.85.

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 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.