**I2C VVC** –Quick Reference

**VVC**

For general information see UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc.

I2C Master (see page 2 for I2C Slave)

|  |
| --- |
| 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, msg, [action\_when\_transfer\_is\_done, [scope]]) |
| 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, [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\_vvc.vhd*

**I2C VVC** –Quick Reference  
I2C Slave (see page 1 for I2C Master)

**VVC**

|  |
| --- |
| 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\_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\_receive (VVCT, vvc\_instance\_idx, num\_bytes, msg, [scope]) |
| Example: i2c\_master\_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**

**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()**

|  |  |  |
| --- | --- | --- |
| **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 |
|  |  |  |

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 |  |

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 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 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 [ ].*

# VVC procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **i2c\_master\_transmit()** | **i2c\_master\_transmit (VVCT, vvc\_instance\_idx, addr, data, msg, [action\_when\_transfer\_is\_done, [scope]])**  The i2c\_master\_transmit() 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 i2c\_master\_transmit() procedure, described in the I2C BFM QuickRef. The i2c\_master\_transmit() 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’.  Examples:  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); |
| **i2c\_master\_check()** | **i2c\_master\_check (VVCT, instance\_idx, addr, data, msg, [action\_when\_transfer\_is\_done, [alert\_level, [scope]]])**  The i2c\_master\_check () 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 i2c\_master\_check() procedure, described in the I2C BFM QuickRef. The received data will not be stored by this procedure. The i2c\_master\_check() 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’.  Examples:  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); |
| **i2c\_master\_receive()** | **i2c\_master\_receive (VVCT, instance\_idx, channel, msg, [scope])**  The i2c\_master\_receive() 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 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).  Example:  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);  **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 := 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"); |
| **i2c\_slave\_transmit()** | **i2c\_slave\_transmit (VVCT, vvc\_instance\_idx, data, msg, [scope])**  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’.  Examples:  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); |
| **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]])**  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’.  Examples:  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); |
| **i2c\_slave\_receive()** | **i2c\_slave\_receive (VVCT, instance\_idx, num\_bytes, msg, [scope])**  See description and fetch\_result() example in the description for i2c\_master\_receive()  Example:  i2c\_slave\_receive(I2C\_VVCT, 1, 1, "One byte from master to slave", C\_SCOPE); |

# VVC Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_I2C\_VVC\_CONFIG\_DEFAULT** | **Description** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_I2C\_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\_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;

# 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** |
| 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 |

# 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.

# Compilation

The I2C VVC must be compiled with VHDL 2008.   
It is dependent on the following libraries

* ***UVVM Utility Library (UVVM-Util), version 2.2.0 and up***
* ***UVVM VVC Framework, version 2.1.0 and up***
* ***I2C BFM***

Before compiling the I2C VVC, make sure that uvvm\_vvc\_framework and uvvm\_util 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 | 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 |

# 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

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**