**RGMII 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.

*rgmii\_vvc.vhd*

|  |
| --- |
| rgmii\_write (VVCT, vvc\_instance\_idx, channel, data\_array, msg, [scope]) |
| Example: rgmii\_write(RGMII\_VVCT, 0, TX, v\_data\_array(0 to v\_numBytes-1), “Write v\_numBytes to DUT”, C\_SCOPE);  Example: rgmii\_write(RGMII\_VVCT, 0, TX, (x”01”, x”02”, x”03”, x”04”), “Write 4 bytes to DUT”); |

|  |
| --- |
| rgmii\_read (VVCT, vvc\_instance\_idx, channel, msg, [scope]) |
| Example: rgmii\_read(RGMII\_VVCT, 1, RX, “Read data which is stored in VVC and will be fetched later using fetch\_result() “); |

|  |
| --- |
| rgmii\_expect (VVCT, vvc\_instance\_idx, channel, data\_exp, msg, [alert\_level, [scope]]) |
| Example: rgmii\_expect(RGMII\_VVCT, 1, RX, v\_data\_array(0 to v\_numBytes-1), “Expect v\_numBytes from DUT”, C\_SCOPE, ERROR);  Example: rgmii\_expect(RGMII\_VVCT, 1, RX, (x”01”, x”02”, x”03”, x”04”), “Expect 4 bytes from DUT”); |



|  |  |
| --- | --- |
| RGMII VVC Configuration record **´vvc\_config´ --** accessible via **shared\_rgmii\_vvc\_config** | **Common VVC procedures applicable for this VVC** |
| |  |  |  | | --- | --- | --- | | **Record element** | **Type** | **C\_RGMII\_VVC\_CONFIG\_DEFAULT** | | inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_RGMII\_INTER\_BFM\_DELAY\_DEFAULT | | cmd\_queue\_count\_max | natural | C\_CMD\_QUEUE\_COUNT\_MAX | | cmd\_queue\_count\_threshold | natural | C\_CMD\_QUEUE\_COUNT\_THRESHOLD | | cmd\_queue\_count\_threshold\_severity | t\_alert\_level | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | | result\_queue\_count\_max | natural | C\_RESULT\_QUEUE\_COUNT\_MAX | | result\_queue\_count\_threshold | natural | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD | | result\_queue\_count\_threshold\_severity | t\_alert\_level | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | | bfm\_config | t\_rgmii\_bfm\_config | C\_RGMII\_BFM\_CONFIG\_DEFAULT | | msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT | |  |  |  | | - See UVVM Methods QuickRef for details. |
| **await\_[any]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()** |
| RGMII VVC Status record signal **´vvc\_status´ --** accessible via **shared\_rgmii\_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 | RGMII\_VVCT | VVC target type compiled into each VVC in order to differentiate between VVCs. |
| vvc\_instance\_idx | integer | 0 | Instance number of the VVC |
| channel | t\_channel | TX, RX | The VVC channel of the VVC instance |

VVC functional parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| data\_array  data\_exp | t\_byte\_array | (x“D0”, x”D1”, x“D2”, x”D3”) | An array of bytes containing the data to be written/read.  data\_array(0) is written/read first, while data\_array(data\_array’high) is written/read last.  For clarity, data\_array is required to be ascending, for example defined by the test sequencer as follows:  variable v\_data\_array : t\_byte\_array(0 to C\_MAX\_BYTES-1); |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the procedure. |
| msg | string | “Write bytes” | A custom message to be appended in the log/alert |
| scope | string | “RGMII\_VVC” | A string describing the scope from which the log/alert originates.  In a simple single sequencer typically "RGMII\_BFM". In a verification component typically "RGMII\_VVC ". |

VVC entity signals

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| rgmii\_vvc\_tx\_if | t\_rgmii\_tx\_if | See RGMII BFM documentation. |
| rgmii\_vvc\_rx\_if | t\_rgmii\_rx\_if | See RGMII BFM documentation. |

VVC entity generic constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| GC\_INSTANCE\_IDX | natural | - | Instance number to assign the VVC. |
| GC\_RGMII\_BFM\_CONFIG | t\_rgmii\_bfm\_config | C\_RGMII\_BFM\_  CONFIG\_DEFAULT | Configuration for the RGMII BFM, see RGMII 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

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **rgmii\_write()** | **rgmii\_write (VVCT, vvc\_instance\_idx, channel, data\_array, msg, [scope])**  The rgmii\_write() VVC procedure adds a write command to the RGMII VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the RGMII BFM rgmii\_write() procedure, described in the RGMII BFM QuickRef. |
| **rgmii\_read()** | **rgmii\_read (VVCT, vvc\_instance\_idx, channel, msg, [scope])**  The rgmii\_read() VVC procedure adds a read command to the RGMII VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the RGMII BFM rgmii\_read() procedure, described in the RGMII BFM QuickRef.  The value received from the DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the received data and metadata will be stored in the VVC for a potential future fetch (see example with *fetch\_result* below).  **Example with fetch\_result() call**: Result is placed in **v\_result**  variable v\_cmd\_idx : natural; -- Command index for the last receive  variable v\_result : work.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from read (data and metadata)  (…)  rgmii\_read(RGMII\_VVCT, 1, RX, “Read data in VVC”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(RGMII\_VVCT, 1, RX);  await\_completion(RGMII\_VVCT, 1, RX, 1 ms, “Wait for read to finish”);  fetch\_result(RGMII\_VVCT, 1, RX, v\_cmd\_idx, **v\_result**, “Fetching result from read operation”); |
| **rgmii\_expect()** | **rgmii\_expect (VVCT, vvc\_instance\_idx, channel, data\_exp, msg, [alert\_level, [scope]])**  The rgmii\_expect() VVC procedure adds an expect command to the RGMII VVC executor queue, which will run as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the RGMII BFM rgmii\_expect() procedure, described in the RGMII BFM QuickRef. |
|  |  |

# VVC Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_RGMII\_VVC\_CONFIG\_DEFAULT** | **Description** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_RGMII\_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\_CMD\_QUEUE\_COUNT\_MAX | 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 initiated if 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\_rgmii\_bfm\_config | C\_RGMII\_BFM\_CONFIG\_DEFAULT | Configuration for RGMII BFM. See quick reference for RGMII 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\_rgmii\_vvc\_config(1).inter\_bfm\_delay.delay\_in\_time := 50 ns;

shared\_rgmii\_vvc\_config(1).bfm\_config.clock\_period := 10 ns;

# VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared\_rgmii\_vvc\_status record from the test sequencer. 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 |

# 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.   
More information can be found in UVVM Essential Mechanisms PDF in the UVVM VVC Framework doc folder.

# 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.

See UVVM VVC Framework Essential Mechanisms PDF, section 6, for additional information.

|  |  |  |  |
| --- | --- | --- | --- |
| **Info field** | **Type** | **Default** | **Description** |
| operation | t\_operation | NO\_OPERATION | Current VVC operation, e.g. INSERT\_DELAY, POLL\_UNTIL, READ, WRITE. |
| data\_array | t\_byte\_array(0 to 299) | (others => (others => '0')) | An array of bytes containing the data to be written/read.  data\_array(0) is written/read first, while data\_array(data\_array’high) is written/read last. |
| 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. |

# Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the TO\_SB parameter in supported method calls. 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 RGMII scoreboard is accessible from the testbench as a shared variable RGMII\_SB, located in the vvc\_methods\_pkg.vhd. All of the listed Generic Scoreboard commands are available for the RGMII VVC using this shared variable.

# VVC Interface

In this VVC, the interface has been encapsulated in two signal records of type *t\_rgmii\_tx\_if* for the signals going to the DUT and *t\_rgmii\_rx\_if* for the signals coming from the DUT in order to improve readability of the code.

# Additional Documentation

Additional documentation about UVVM and its features can be found under “/uvvm\_vvc\_framework/doc/”.

For additional documentation on the RGMII standard, see the specification “Reduced Gigabit Media Independent Interface (RGMII) Version 2.0”.

# Compilation

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

* ***UVVM Utility Library (UVVM-Util), version 2.12.0 and up***
* ***UVVM VVC Framework, version 2.7.3 and up***
* ***RGMII BFM***
* ***Bitvis VIP Scoreboard***

Before compiling the RGMII VVC, assure 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 RGMII VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_rgmii | rgmii\_bfm\_pkg.vhd | RGMII BFM |
| bitvis\_vip\_rgmii | transaction\_pkg.vhd | RGMII transaction package with DTT types, constants etc. |
| bitvis\_vip\_rgmii | vvc\_cmd\_pkg.vhd | RMGII VVC command types and operations |
| bitvis\_vip\_rgmii | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the RGMII VVC library. |
| bitvis\_vip\_rgmii | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | UVVM framework common methods compiled into the RGMII VVC library |
| bitvis\_vip\_rgmii | vvc\_methods\_pkg.vhd | RGMII VVC methods |
| bitvis\_vip\_rgmii | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_rgmii | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the RGMII VVC library |
| bitvis\_vip\_rgmii | rgmii\_tx\_vvc.vhd | RGMII TX VVC |
| bitvis\_vip\_rgmii | rgmii\_rx\_vvc.vhd | RGMII RX VVC |
| bitvis\_vip\_rgmii | rgmii\_vvc.vhd | RGMII VVC |

# 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 RGMII. The given VIP complies with the basic RGMII protocol and thus allows a normal access towards an RGMII interface. This VIP is not RGMII protocol checker. For a more advanced VIP please contact Bitvis AS at [support@bitvis.no](mailto: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**