**RGMII BFM** –Quick Reference

**BFM**

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

|  |
| --- |
| rgmii\_write (data\_array, msg, rgmii\_tx\_if, [scope, [msg\_id\_panel, [config]]]) |
| Example: rgmii\_write(v\_data\_array(0 to v\_numBytes-1), "Write v\_numBytes bytes", rgmii\_tx\_if, C\_SCOPE, shared\_msg\_id\_panel, rgmii\_bfm\_config); Example: rgmii\_write((x”01”, x”02”, x”03”, x”04”), "Write 4 bytes", rgmii\_tx\_if); |

*rgmii\_bfm\_pkg.vhd*

|  |
| --- |
| rgmii\_read (data\_array, data\_len, msg, rgmii\_rx\_if, [scope, [msg\_id\_panel, [config, [ext\_proc\_call]]]]) |
| Example: rgmii\_read(v\_data\_array, v\_numBytes, "Read v\_numBytes bytes", rgmii\_rx\_if, C\_SCOPE, shared\_msg\_id\_panel, rgmii\_bfm\_config, “rgmii\_expect()”);  Example: rgmii\_read(v\_data\_array, v\_numBytes, "Read v\_numBytes bytes", rgmii\_rx\_if); |

|  |
| --- |
| rgmii\_expect (data\_exp, msg, rgmii\_rx\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]]) |
| Example: rgmii\_expect(v\_data\_array(0 to v\_numBytes-1), "Expect v\_numBytes bytes", rgmii\_rx\_if, ERROR, C\_SCOPE, shared\_msg\_id\_panel, rgmii\_bfm\_config); Example: rgmii\_expect((x”01”, x”02”, x”03”, x”04”), "Expect 4 bytes", rgmii\_rx\_if); |

|  |
| --- |
| init\_rgmii\_if\_signals (VOID) |
| Example: rgmii\_tx\_if <= init\_rgmii\_if\_signals(VOID); |



|  |  |
| --- | --- |
| **Record element** | **Type** |
| txc | std\_logic |
| txd | std\_logic\_vector |
| tx\_ctl | std\_logic |

BFM Configuration record ‘**t\_rgmii\_bfm\_config’** Signal record **‘t\_rgmii\_tx\_if’**

|  |  |  |
| --- | --- | --- |
| Record element | Type | C\_RGMII\_BFM\_CONFIG\_DEFAULT |
| max\_wait\_cycles | integer | 10 |
| max\_wait\_cycles\_severity | t\_alert\_level | ERROR |
| clock\_period | time | -1 ns |
| rx\_clock\_skew | time | -1 ns |
| id\_for\_bfm | t\_msg\_id | ID\_BFM |

Signal record **‘t\_rgmii\_rx\_if’**

|  |  |
| --- | --- |
| **Record element** | **Type** |
| rxc | std\_logic |
| rxd | std\_logic\_vector |
| rx\_ctl | std\_logic |

BFM signal parameters

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| txc | std\_logic | TX reference clock |
| txd | std\_logic\_vector | TX data lines (to DUT) |
| tx\_ctl | std\_logic | TX enable |
| rxc | std\_logic | RX reference clock |
| rxd | std\_logic\_vector | RX data lines (from DUT) |
| rx\_ctl | std\_logic | RX enable |

Note: tx\_ctl & rx\_ctl only represent TXEN & RXEN respectively, the functionality of TXERR & RXERR is not implemented.

Also, there is no support for RGMII-ID (use of Tsetup & Thold). For more information see the specification “Reduced Gigabit Media Independent Interface (RGMII) Version 2.0”.

BFM non-signal 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); |
| data\_len | natural | v\_data\_len | The number of valid bytes in the data\_array. Note that the data\_array can be bigger and that is why the length is returned. |
| 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\_BFM" | 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 ". |
| msg\_id\_panel | t\_msg\_id\_panel | shared\_msg\_id\_panel | Optional msg\_id\_panel, controlling verbosity within a specified scope. Defaults to a common message ID panel defined in the UVVM-Util adaptations package. |
| config | t\_rgmii\_bfm\_config | C\_RGMII\_BFM\_  CONFIG\_DEFAULT | Configuration of BFM behaviour and restrictions. See section 2 for details. |
| ext\_proc\_call | string | “rgmii\_expect()” | External procedure call. Only use when called from another BFM procedure. |

BFM details

# BFM procedure details

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **rgmii\_write()** | **rgmii\_write (data\_array, msg, rgmii\_tx\_if, [scope, [msg\_id\_panel, [config]]])**  The rgmii\_write() procedure writes 4 bits of data on each clock edge. The bits 3:0 are written on the rising edge and the bits 7:4 on the falling edge.  The length and data are defined by the “data\_array” argument, which is a t\_byte\_array.  data\_array(0) is written first, while data\_array(data\_array’high) is written last. |
| **rgmii\_read()** | **rgmii\_read (data\_array, data\_len, msg, rgmii\_rx\_if, [scope, [msg\_id\_panel, [config, [ext\_proc\_call]]]])**  The rgmii\_read() procedure reads 4 bits of data on each clock edge. The bits 3:0 are read on the rising edge and the bits 7:4 on the falling edge. To avoid having to delay the receiver’s clock, the config rx\_clock\_skew is used to set the sampling time of the data.  The received data is stored in the data\_array output, which is a t\_byte\_array. The number of valid bytes in the data\_array is stored in data\_len.  data\_array(0) is read first, while data\_array(data\_array’high) is read last. |
| **rgmii\_expect()** | **rgmii\_expect (data\_exp, msg, rgmii\_rx\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]])**    Calls the rgmii\_read() procedure, then compares the received data with data\_exp. |
| **init\_rgmii\_if\_signals()** | **init\_rgmii\_if\_signals(VOID)**  This function initializes the RGMII interface. All the BFM outputs are set to zeros ('0') |

# BFM Configuration record

Type name: t\_rgmii\_bfm\_config

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_RGMII\_BFM\_CONFIG\_DEFAULT** | **Description** |
| max\_wait\_cycles | integer | 10 | Used for setting the maximum cycles to wait before an alert is issued when waiting for signals from the DUT. |
| max\_wait\_cycles\_severity | t\_alert\_level | ERROR | Severity if max\_wait\_cycles expires. |
| clock\_period | time | -1 ns | Period of the clock signal. |
| rx\_clock\_skew | time | -1 ns | Skew of the sampling of the data in connection to the RX clock edges. Suggested value is clock\_period/4. |
| id\_for\_bfm | t\_msg\_id | ID\_BFM | The message ID used as a general message ID in the BFM. |
|  |  |  |  |

# Compilation

The RGMII BFM may only be compiled with VHDL 2008. It is dependent on the UVVM Utility Library (UVVM-Util), which is only compatible with VHDL 2008.

See the separate UVVM-Util documentation for more info. After UVVM-Util has been compiled, the rgmii\_bfm\_pkg.vhd BFM can be compiled into any desired library.

See UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc for information about compile scripts.

## Simulator compatibility and setup

See README.md for a list of supported simulators. For required simulator setup see UVVM-Util Quick reference.

# Local BFM overloads

A good approach for better readability and maintainability is to make simple, local overloads for the BFM procedures in the TB process.

This allows calling the BFM procedures with the key parameters only

e.g.

rgmii\_write(v\_data\_array(0 to 1), "msg");

rather than

rgmii\_write(v\_data\_array(0 to 1), "msg", rgmii\_tx\_if, C\_SCOPE, shared\_msg\_id\_panel, rgmii\_bfm\_config);

By defining the local overload as e.g.:

procedure rgmii\_write(

constant data\_array : in t\_byte\_array;

constant msg : in string) is

begin

rgmii\_write(data\_array, -- keep as is  
 msg, -- keep as is

clk, -- Clock signal

rgmii\_tx\_if, -- Signal must be visible in local process scope

C\_SCOPE, -- Just use the default

shared\_msg\_id\_panel, -- Use global, shared msg\_id\_panel

C\_RGMII\_BFM\_CONFIG\_LOCAL); -- Use locally defined configuration or C\_RGMII\_BFM\_CONFIG\_DEFAULT

end;

Using a local overload like this also allows the following – if wanted:

* Set up defaults for constants. May be different for two overloads of the same BFM
* Apply dedicated message\_id\_panel to allow dedicated verbosity control

IMPORTANT

This is a simplified Bus Functional Model (BFM) for RGMII. The given BFM complies with the basic RGMII protocol and thus allows a normal access towards an RGMII interface. This BFM is not RGMII protocol checker. For a more advanced BFM 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**