**GMII BFM** –Quick Reference

**BFM**

**NOTE: As of UVVM v3.x, all shared variables have been made protected. This means that any access to shared variables must be done**

**using get- and set-methods. This documentation has not yet been updated with the methods for accessing these variables, but will be very soon.**

This is a stripped-down version of GMII with only data lines.

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

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

*gmii\_bfm\_pkg.vhd*

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

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

|  |
| --- |
| init\_gmii\_if\_signals (VOID) |
| Example: gmii\_tx\_if <= init\_gmii\_if\_signals(VOID); |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | BFM Configuration record ´**t\_gmii\_bfm\_config´** | | | | **Record element** | **Type** | **C\_GMII\_BFM\_CONFIG\_DEFAULT** | | max\_wait\_cycles | integer | 10 | | max\_wait\_cycles\_severity | t\_alert\_level | ERROR | | clock\_period | time | -1 ns | | clock\_period\_margin | time | 0 ns | | clock\_margin\_severity | t\_alert\_level | TB\_ERROR | | setup\_time | time | -1 ns | | hold\_time | time | -1 ns | | bfm\_sync | t\_bfm\_sync | SYNC\_ON\_CLOCK\_ONLY | | match\_strictness | t\_match\_strictness | MATCH\_EXACT | | id\_for\_bfm | t\_msg\_id | ID\_BFM | | |  |  | | --- | --- | | Signal record ‘**t\_gmii\_tx\_if’** | | | **Record element** | **Type** | | gtxclk | std\_logic | | txd | std\_logic\_vector(7 downto 0) | | txen | std\_logic |  |  |  | | --- | --- | | Signal record ‘**t\_gmii\_rx\_if’** | | | **Record element** | **Type** | | rxclk | std\_logic | | rxd | std\_logic\_vector(7 downto 0) | | rxdv | std\_logic | |

BFM signal parameters

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| gtxclk | std\_logic | TX reference clock |
| txd | std\_logic\_vector | TX data lines (to DUT) |
| txen | std\_logic | TX enable |
| rxclk | std\_logic | RX reference clock |
| rxd | std\_logic\_vector | RX data lines (from DUT) |
| rxdv | std\_logic | RX data valid |

BFM non-signal parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| data\_array  data\_exp | t\_slv\_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\_slv\_array(0 to C\_MAX\_BYTES-1)(7 downto 0); |
| 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 | "GMII\_BFM" | A string describing the scope from which the log/alert originates. In a simple single sequencer typically "GMII\_BFM". In a verification component typically "GMII\_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\_gmii\_bfm\_config | C\_GMII\_BFM\_CONFIG\_DEFAULT | Configuration of BFM behaviour and restrictions. See section 0 for details. |
| ext\_proc\_call | string | “gmii\_expect()” | External procedure call. Only use when called from another BFM procedure. |

BFM details

# BFM procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **gmii\_write()** | **gmii\_write (data\_array, msg, gmii\_tx\_if, [scope, [msg\_id\_panel, [config]]])**  The gmii\_write() procedure writes data to the DUT.  The length and data are defined by the “data\_array” argument, which is a t\_slv\_array.  data\_array(0) is written first, while data\_array(data\_array’high) is written last. |
| **gmii\_read()** | **gmii\_read (data\_array, data\_len, msg, gmii\_rx\_if, [scope, [msg\_id\_panel, [config, ext\_proc\_call]]]])**  The gmii\_read() procedure reads data from the DUT.  The received data is stored in the data\_array output, which is a t\_slv\_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. |
| **gmii\_expect()** | **gmii\_expect (data\_exp, msg, gmii\_rx\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]])**  Calls the gmii\_read() procedure, then compares the received data with data\_exp. |
| **init\_gmii\_if\_signals()** | **init\_gmii\_if\_signals (VOID)**  This function initializes the GMII interface. All the BFM outputs are set to zeros ('0') |

# BFM Configuration record

Type name: t\_gmii\_bfm\_config

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_GMII\_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. |
| clock\_period\_margin | time | 0 ns | Input clock period margin to specified clock\_period. |
| clock\_margin\_severity | t\_alert\_level | TB\_ERROR | The above margin will have this severity. |
| setup\_time | time | -1 ns | Setup time for generated signals. Suggested value is clock\_period/4.  An alert is reported if setup\_time exceed clock\_period/2. |
| hold\_time | time | -1 ns | Hold time for generated signals. Suggested value is clock\_period/4.  An alert is reported if hold\_time exceed clock\_period/2. |
| bfm\_sync | t\_bfm\_sync | SYNC\_ON\_CLOCK\_ONLY | When set to SYNC\_ON\_CLOCK\_ONLY the BFM will enter on the first falling edge, estimate the clock period, synchronise the output signals and exit ¼ clock period after a succeeding rising edge. When set to SYNC\_WITH\_SETUP\_AND\_HOLD the BFM will use the configured setup\_time, hold\_time and clock\_period to synchronise output signals with clock edges. |
| match\_strictness | t\_match\_strictness | MATCH\_EXACT | Matching strictness for std\_logic values in check procedures.  MATCH\_EXACT requires both values to be the same. Note that the expected value  can contain the don’t care operator ‘-‘.  MATCH\_STD allows comparisons between ‘H’ and ‘1’, ‘L’ and ‘0’ and ‘-‘ in both values. |
| id\_for\_bfm | t\_msg\_id | ID\_BFM | The message ID used as a general message ID in the BFM. |

# Compilation

The GMII 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 gmii\_bfm\_pkg.vhd 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.

gmii\_write(v\_data\_array(0 to 1), “msg”);

rather than

gmii\_write(v\_data\_array(0 to 1), “msg”, gmii\_tx\_if, C\_SCOPE, shared\_msg\_id\_panel, C\_GMII\_BFM\_CONFIG\_DEFAULT);

By defining the local overload as e.g.:

procedure gmii\_write(

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

constant msg : in string) is

begin

gmii\_write(data\_array, -- keep as is

msg, -- keep as is

gmii\_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\_GMII\_BFM\_CONFIG\_LOCAL); -- Use locally defined configuration or C\_GMII\_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 GMII. The given BFM complies with the basic GMII protocol and thus allows a normal access towards a GMII interface. This BFM is not a GMII protocol checker. For a more advanced BFM 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**