**Avalon-MM BFM** –Quick Reference

**BFM**

**BFM**

|  |
| --- |
| avalon\_mm\_write (addr\_value, data\_value, msg, clk, avalon\_mm\_if, [byte\_enable], [scope, [msg\_id\_panel, [config]]]) |
| Example: avalon\_mm\_write(x"11005500", x”AAFF0055”, “Writing test to Peripheral 1”, clk, avalon\_mm\_if); -- Without byte\_enable  Example: avalon\_mm\_write(x"11005500", x”AAFF0055”, “Writing test to Peripheral 1”, clk, avalon\_mm\_if, “1111”); -- With byte\_enable  *Suggested usage: avalon\_mm\_write(C\_ADDR\_DMA, x”AAFF0055”, “Writing data to DMA”); -- Suggested usage requires local overload (see section 5)* |

*avalon\_mm\_bfm\_pkg.vhd*

|  |
| --- |
| avalon\_mm\_read (addr\_value, data\_value, msg, clk, avalon\_mm\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]]) |
| Example: avalon\_mm\_read(x"11355000", v\_data\_out, “Read from Peripheral 1”, clk, avalon\_mm\_if);  *Suggested usage: avalon\_mm\_read(C\_ADDR\_IO, v\_data\_out, “Read from IO device”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| avalon\_mm\_check (addr\_value, data\_exp, msg, clk, avalon\_mm\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]]) |
| Example: avalon\_mm\_check(x"6840A000", x”00443B16”, *“Check data from Peripheral 1”*, clk, avalon\_mm\_if);  *Suggested usage: avalon\_mm\_check(C\_ADDR\_IO, x”00443B16”, “Check data from IO device”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| avalon\_mm\_reset (clk, avalon\_mm\_if, num\_rst\_cycles, msg, [scope, [msg\_id\_panel, [config]]]) |
| Example: avalon\_mm\_reset(clk, avalon\_mm\_if, 5, “Resetting Avalon MM Interface”);  *Suggested usage: avalon\_mm\_check(C\_NUM\_RST\_CYCLES, “Resetting Avalon MM Interface”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| init\_avalon\_mm\_if\_signals (addr\_width, data\_width, [lock\_value]) |
| Example: avalon\_mm\_if <= init\_avalon\_mm\_to\_dut\_signals(addr\_width, data\_width); |

Signal record ´**t\_avalon\_mm\_if´**

|  |  |
| --- | --- |
| **Record element** | **Type** |
| reset | std\_logic |
| address | std\_logic\_vector |
| begintransfer | std\_logic |
| byte\_enable | std\_logic\_vector |
| chipselect | std\_logic |
| write | std\_logic |
| writedata | std\_logic\_vector |
| read | std\_logic |
| lock | std\_logic |
| readdata | std\_logic\_vector |
| response | std\_logic\_vector |
| waitrequest | std\_logic |
| readdatavalid | std\_logic |
| irq | std\_logic |

BFM Configuration record ´**t\_avalon\_mm\_bfm\_config´**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT** |
| max\_wait\_cycles | integer | 10 |
| max\_wait\_cycles\_severity | t\_alert\_level | TB\_FAILURE |
| 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 |
| num\_wait\_states\_read | natural | 0 |
| num\_wait\_states\_write | natural | 0 |
| use\_waitrequest | boolean | true |
| use\_readdatavalid | boolean | false |
| use\_response\_signal | boolean | true |
| use\_begintransfer | boolean | false |
| id\_for\_bfm | t\_msg\_id | ID\_BFM |
| id\_for\_bfm\_wait | t\_msg\_id | ID\_BFM\_WAIT |
| id\_for\_bfm\_poll | t\_msg\_id | ID\_BFM\_POLL |

Advanced Avalon-MM commands

|  |
| --- |
| avalon\_mm\_lock (avalon\_mm\_if, msg, [scope, [msg\_id\_panel, [config]]]) |
| Example: avalon\_mm\_lock(avalon\_mm\_if “Locking Avalon MM Bus”); |

|  |
| --- |
| avalon\_mm\_unlock (avalon\_mm\_if, msg, [scope, [msg\_id\_panel, [config]]]) |
| Example: avalon\_mm\_unlock(avalon\_mm\_if “Unlocking Avalon MM Bus”); |

|  |
| --- |
| avalon\_mm\_read\_request (addr\_value, msg, clk, avalon\_mm\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]]) |
| Example: avalon\_mm\_read\_request(x"11355000", “Start read from Peripheral 1”, clk, avalon\_mm\_if);  *Suggested usage: avalon\_mm\_read\_request(C\_ADDR\_IO, “Start read from IO device”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| avalon\_mm\_read\_response (addr\_value, data\_value, msg, clk, avalon\_mm\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]]) |
| Example: avalon\_mm\_read\_response(x"11355000", v\_data\_out, “Get read response from Peripheral 1”, clk, avalon\_mm\_if);  *Suggested usage: avalon\_mm\_read\_response(C\_ADDR\_IO, v\_data\_out, “Get read response from IO device”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| avalon\_mm\_check\_response (addr\_value, data\_value, msg, clk, avalon\_mm\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]]) |
| Example: avalon\_mm\_check\_response(x"6840A000", x”00443B16”, *“Check data from Peripheral 1”*, clk, avalon\_mm\_if);  *Suggested usage: avalon\_mm\_check\_response(C\_ADDR\_IO, x”00443B16”, “Check data from IO device”); -- Suggested usage requires local overload (see section 5)* |

BFM non-signal parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| addr\_value | unsigned | x”125A” | The address of an Avalon-MM accessible register. |
| data\_value | std\_logic\_vector | x”20D3” | The data value to be written to the addressed register |
| data\_exp | std\_logic\_vector | x”0D” | The data value to expect when reading the addressed register. A mismatch results in an alert ‘alert\_level’ |
| byte\_enable | std\_logic\_vector | x”11” | This argument selects which bytes to use (all ‘1’ means all bytes are updated) |
| lock\_value | std\_logic | ‘0’ | init\_avalon\_mm\_if\_signals argument for deciding the value of the lock signal. Default ‘0’, Only used by internal BFM procedures. |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the procedure. |
| msg | string | “Set state active on peripheral 1” | A custom message to be appended in the log/alert. |
| scope | string | "AVALON MM BFM" | A string describing the scope from which the log/alert originates. In a simple single sequencer typically "AVALON MM BFM". In a verification component typically "AVALON\_MM\_VVC ". |
| msg\_id\_panel | t\_msg\_id\_panel | shared\_msg\_id\_panel | Optional message ID panel, controlling verbosity within a specified scope. Defaults to a common ID panel defined in the UVVM-Util adaptations package. |
| config | t\_avalon\_mm\_bfm\_config | C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT | Configuration of BFM behaviour and restrictions. See section 0 for details. |

BFM signal parameters

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| clk | std\_logic | The clock signal used to read and write data in/out of Avalon-MM BFM. |
| avalon\_mm\_if | t\_avalon\_mm\_if | See table “Signal record ‘t\_avalon\_mm\_if” |

Note: All signals are active high. See Avalon MM documentation for protocol description.

For more information on the Avalon MM signals, please see the Avalon MM specification.

BFM details

# BFM procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **avalon\_mm\_write()** | **avalon\_mm\_write(addr\_value, data\_value, msg, clk, avalon\_mm\_if, [byte\_enable,] [scope, [msg\_id\_panel, [config]]])**  The avalon\_mm\_write() procedure writes the given data to the given address of the DUT, using the Avalon-MM protocol. For protocol details, see the Avalon-MM specification.   * If the byte\_enable argument is not used, it will be set to all ‘1’, i.e. all bytes are used. * The avalon\_mm\_write() procedure supports wait-request or fixed wait-states, but not both. If ‘config.use\_waitrequest’ is set to false, ‘config. num\_wait\_states’ will be used as the number of cycles to use as fixed wait cycles. * The default value of scope is C\_SCOPE (“AVALON MM BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM-Util. * The default value of config is C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT, see table on the first page. * A log message is written after procedure completes if ID\_BFM ID is enabled for the specified message ID panel.   The procedure reports an alert if:   * waitrequest is enabled for more than ‘config.max\_wait\_cycles’ clock cycles (alert level: ‘config.max\_wait\_cycles\_severity’)     Examples:  avalon\_mm\_write(x"11005500", x”AAFF0055”, “Writing test to Peripheral 1”, clk, avalon\_mm\_if, C\_SCOPE, shared\_msg\_id\_panel,   C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  avalon\_mm\_write(x"11005500", x”AAFF0055”, “Writing test to Peripheral 1”, clk, avalon\_mm\_if, “1111”, C\_SCOPE, shared\_msg\_id\_panel,   C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_write(C\_ADDR\_DMA, x”AAFF0055”, “Writing data to DMA”); |
| **avalon\_mm\_read()** | **avalon\_mm\_read(addr\_value, data\_value, msg, clk, avalon\_mm\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]])**  The avalon\_mm\_read() procedure reads data from the given address of the DUT, using the Avalon-MM protocol. For protocol details, see the Avalon-MM specification. The read data is placed on the output ‘data\_value’ when the read has completed.   * The avalon\_mm\_read() procedure supports pipelining/fixed wait-states, readdatavalid and/or waitrequest, set by the config parameter.   + The maximum number of wait cycles while waiting for readdatavalid is given in ‘config.max\_wait\_cycles’   + The maximum number of cycles acceptable to be stalled by waitrequest is given in ‘config.max\_wait\_cycles’   + If use\_waitrequest and use\_readdatavalid are disabled in the config, the read procedure will use the num\_wait\_states as readWaitTime. * The default value of scope is C\_SCOPE (“AVALON MM BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM-Util. * The default value of config is C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT, see table on the first page. * The default value of proc\_name is “avalon\_mm\_read”. This argument is intended to be used internally, when procedure is called by avalon\_mm\_check(). * A log message is written if ID\_BFM ID is enabled for the specified message ID panel. This will only occur if the argument proc\_name is left unchanged. * The BFM can be configured to use waitrequest and readdatavalid in the config parameter.   The procedure reports an alert if:   * waitrequest is enabled for more than ‘config.max\_wait\_cycles’ clock cycles (alert level: ‘config.max\_wait\_cycles\_severity’) * readdatavalid is not set active for more than ‘config.max\_wait\_cycles’ clock cycles (alert level: ‘config.max\_wait\_cycles\_severity’)     Example:  avalon\_mm\_read(x“5A001120”, v\_data\_out, “Read from Peripheral 1”, clk, avalon\_mm\_if, C\_SCOPE, shared\_msg\_id\_panel,   C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_read(C\_ADDR\_IO, v\_data\_out, “Reading from IO device”); |
| **avalon\_mm\_check()** | **avalon\_mm\_check(addr\_value, data\_exp, msg, clk, avalon\_mm\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]])**  The avalon\_mm\_check() procedure reads data from the given address of the DUT, using the Avalon-MM protocol. For protocol details, see the Avalon-MM specification. After reading data from the Avalon-MM bus, the read data is compared with the expected data, ‘data\_exp’.   * The default value of alert\_level is ERROR * The default value of scope is C\_SCOPE (“AVALON MM BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM\_Util. * The default value of config is C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT, see table on the first page. * If the check was successful, and the read data matches the expected data, a log message is written with ID ID\_BFM (if this ID has been enabled). * If the read data did not match the expected data, an alert with severity ‘alert\_level’ will be reported.   The procedure also report alerts for the same conditions as the avalon\_mm\_read() procedure.  Example:  avalon\_mm\_check(x”11AA5100”, x”5500133B”, “Check data from Peripheral 1”, clk, avalon\_mm\_if, ERROR, shared\_msg\_id\_panel,   C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_check(C\_ADDR\_UART\_RX, x”55”, “Check data from UART RX buffer”); |
| **avalon\_mm\_reset()** | **avalon\_mm\_reset(clk, avalon\_mm\_if, num\_rst\_cycles, msg, [scope, [msg\_id\_panel, [config]]])**  The avalon\_mm\_reset() procedure resets the avalon\_mm\_if interface by first setting the signals to their default state with init\_avalon\_mm\_if\_signals(), then setting reset active. The reset signal is held active for ‘num\_rst\_cycles’ clock cycles.  A log with ID ID\_BFM is written to the transcript if this ID has been enabled for this message ID panel.  Example:  avalon\_mm\_reset(clk, avalon\_mm\_if, 5, "Resetting Avalon MM Interface", C\_SCOPE, shared\_msg\_id\_panel,   AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_reset(5, “Resetting Avalon MM Interface); |
| **init\_avalon\_mm\_if\_signals()** | **init\_avalon\_mm\_if\_signals(addr\_width, data\_width, [lock\_value])**  This function initializes the Avalon-MM interface. All data and active high BFM outputs are set to ‘0’ and all BFM inputs are set to 'Z'. The value of the lock signal can be specified in the lock\_value argument. This value is default set to ‘0’.  Examples:  avalon\_mm\_if <= init\_avalon\_mm\_if\_signals(addr\_width, data\_width);  avalon\_mm\_if <= init\_avalon\_mm\_if\_signals(addr\_width, data\_width, ‘1’); |
| **avalon\_mm\_lock()** | **avalon\_mm\_lock(avalon\_mm\_if, msg, [scope, [msg\_id\_panel, [config]]])**  The avalon\_mm\_lock() procedure locks the Avalon-MM interface by setting the avalon\_mm\_if signal “lock” to ‘1’. The lock signal will be kept at ‘1’ until avalon\_mm\_unlock() is called.  A log with ID config.id\_for\_bfm is written to the transcript if this ID has been enabled for this message ID panel.  Example:  avalon\_mm\_lock(avalon\_mm\_if, "Locking Avalon MM Interface", C\_SCOPE, shared\_msg\_id\_panel, AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_lock(“Locking Avalon MM Interface); |
| **avalon\_mm\_unlock()** | **avalon\_mm\_unlock(avalon\_mm\_if, msg, [scope, [msg\_id\_panel, [config]]])**  The avalon\_mm\_unlock() procedure unlocks the Avalon-MM interface by setting the avalon\_mm\_if signal “lock” to ‘0’. A log with ID config.id\_for\_bfm is written to the transcript if this ID has been enabled for this message ID panel.  Example:  avalon\_mm\_unlock(avalon\_mm\_if, "Unlocking Avalon MM Interface", C\_SCOPE, shared\_msg\_id\_panel, AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_unlock(“Unlocking Avalon MM Interface); |
| **avalon\_mm\_read\_request()** | **avalon\_mm\_read\_request(addr\_value, msg, clk, avalon\_mm\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]])**  The avalon\_mm\_read\_request() procedure initiates a read request to the given address of the DUT, using the Avalon-MM protocol. For protocol details, see the Avalon-MM specification. This procedure returns as soon as the request has been completed, and will therefore not return any data. This procedure is meant to be used for pipelined reads where multiple read requests can be issued before the slave DUT responds with the read data. The avalon\_mm\_read\_request procedure corresponds to the first half of the avalon\_mm\_read and avalon\_mm\_check procedure. For more information, please see the avalon\_mm\_read procedure description.  The procedure reports an alert if:   * See avalon\_mm\_read procedure     Example:  avalon\_mm\_read\_request(x“5A001120”, “Initiating read from Peripheral 1”, clk, avalon\_mm\_if, C\_SCOPE, shared\_msg\_id\_panel,   C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_read\_request(C\_ADDR\_IO, “Initiating read from IO device”); |
| **avalon\_mm\_read\_response()** | **avalon\_mm\_read\_response(addr\_value, data\_value, msg, clk, avalon\_mm\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]])**  The avalon\_mm\_read\_response() procedure reads data which is returned from the slave DUT, using the Avalon-MM protocol. This procedure is meant as the second half of the avalon\_mm\_read procedure, which is responsible for receiving data that has been requested by the avalon\_mm\_read\_request procedure. For protocol details, see the Avalon-MM specification. The read data is placed on the output ‘data\_value’ when the read has completed. For more information, please see the avalon\_mm\_read procedure description.  The procedure reports an alert if:   * See avalon\_mm\_read procedure     Example:  avalon\_mm\_read\_response(x“5A001120”, v\_data\_out, “Read response from Peripheral 1”, clk, avalon\_mm\_if, C\_SCOPE,   shared\_msg\_id\_panel, C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_read\_response(C\_ADDR\_IO, v\_data\_out, “Reading response from IO device”); |
| **avalon\_mm\_check\_response()** | **avalon\_mm\_check\_response(addr\_value, data\_exp, msg, clk, avalon\_mm\_if,[ alert\_level, [scope, [msg\_id\_panel, [config]]]])**  The avalon\_mm\_check\_response() procedure reads data which is returned from the slave DUT using the Avalon-MM protocol, and compares it to the data in data\_exp. This procedure is meant as the second half of the avalon\_mm\_check procedure, which is responsible for receiving data that has been requested by the avalon\_mm\_read\_request procedure. For protocol details, see the Avalon-MM specification. For more information, please see the avalon\_mm\_check procedure description.  The procedure reports an alert if:   * See avalon\_mm\_check procedure     Example:  avalon\_mm\_check\_response(x“5A001120”, x”5500133B”, “Check response from Peripheral 1”, clk, avalon\_mm\_if, ERROR, C\_SCOPE,   shared\_msg\_id\_panel, C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  avalon\_mm\_check\_response(C\_ADDR\_IO, x”5500133B”, “Checking response from IO device”); |

# 

# BFM Configuration record

Type name: t\_avalon\_mm\_bfm\_config

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT** | **Description** |
| max\_wait\_cycles | integer | 10 | Sets the maximum number of wait cycles before an alert occurs when waiting for readdatavalid or stalling because of waitrequest |
| max\_wait\_cycles\_severity | t\_alert\_level | TB\_FAILURE | The above timeout will have this severity |
| 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\_period\_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 clk\_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 clk\_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. |
| num\_wait\_states\_read | natural | 0 | Number of fixed wait states to use for read |
| num\_wait\_states\_write | natural | 0 | Number of fixed wait states to use for write |
| use\_waitrequest | boolean | true | Set to true if slave uses waitrequest |
| use\_readdatavalid | boolean | false | Set to true if slave uses readdatavalid |
| use\_response\_signal | boolean | true | Whether or not to check the response signal on read |
| use\_begintransfer | boolean | false | Whether or not to use the begintransfer signal. |
| id\_for\_bfm | t\_msg\_id | ID\_BFM | The message ID used as a general message ID in the Avalon BFM |
| id\_for\_bfm\_wait | t\_msg\_id | ID\_BFM\_WAIT | The message ID used for logging waits in the Avalon BFM |
| id\_for\_bfm\_poll | t\_msg\_id | ID\_BFM\_POLL | The message ID used for logging polling in the Avalon BFM |

# Additional Documentation

For additional documentation on the Avalon-MM standard, please see the Avalon specification “Avalon Interface Specifications, MNL-AVABUSREF”, available from Altera.

# Compilation

The Avalon-MM 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 avalon\_mm\_bfm\_pkg.vhd BFM can be compiled into any desired library.

See the 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.

avalon\_mm\_write(C\_ADDR\_PERIPHERAL\_1, C\_TEST\_DATA, “Writing data to Peripheral 1”);

rather than

avalon\_mm\_write(C\_ADDR\_PERIPHERAL\_1, C\_TEST\_DATA, “Writing data to Peripheral 1”, clk, avalon\_mm\_if, C\_SCOPE,   
 shared\_msg\_id\_panel, C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT);

By defining the local overload as e.g.:

procedure avalon\_mm\_write(

constant addr\_value : in unsigned;

constant data\_value : in std\_logic\_vector;

constant msg : in string) is

begin

avalon\_mm\_write(addr\_value, -- keep as is  
 data\_value, -- keep as is

msg, -- keep as is

clk, -- Clock signal

avalon\_mm\_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\_AVALON\_MM\_BFM\_CONFIG\_LOCAL); -- Use locally defined configuration or C\_AVALON\_MM\_BFM\_CONFIG\_DEFAULT

end;

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

* Have address value as natural – and convert in the overload
* 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 Avalon-MM.  
The given BFM complies with the basic Avalon-MM protocol and thus allows a normal access towards an Avalon-MM interface. This BFM is not an Avalon-MM 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**