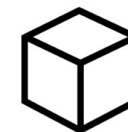


# Avalon-MM BFM – Quick Reference

For general information see UVVM Essential Mechanisms located in `uvvm_vvc_framework/doc`.

**BFM**



`avalon_mm_bfm_pkg.vhd`

**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\_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_reset(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);`

BFM Configuration record `'t_avalon_mm_bfm_config'`

Record element	Type	C_AVALON_MM_BFM_CONFIG_DEFAULT
<code>max_wait_cycles</code>	integer	10
<code>max_wait_cycles_severity</code>	<code>t_alert_level</code>	TB_FAILURE
<code>clock_period</code>	time	10 ns
<code>clock_period_margin</code>	time	0 ns
<code>clock_margin_severity</code>	<code>t_alert_level</code>	TB_ERROR
<code>setup_time</code>	time	2.5 ns
<code>hold_time</code>	time	2.5 ns
<code>num_wait_states_read</code>	natural	0
<code>num_wait_states_write</code>	natural	0
<code>use_waitrequest</code>	boolean	true
<code>use_readdatavalid</code>	boolean	false
<code>use_response_signal</code>	boolean	true
<code>use_begintransfer</code>	boolean	false
<code>id_for_bfm</code>	<code>t_msg_id</code>	ID_BFM
<code>id_for_bfm_wait</code>	<code>t_msg_id</code>	ID_BFM_WAIT
<code>id_for_bfm_poll</code>	<code>t_msg_id</code>	ID_BFM_POLL

Signal record `'t_avalon_mm_if'`

Record element	Type
<code>reset</code>	<code>std_logic</code>
<code>address</code>	<code>std_logic_vector</code>
<code>begintransfer</code>	<code>std_logic</code>
<code>byte_enable</code>	<code>std_logic_vector</code>
<code>chipselect</code>	<code>std_logic</code>
<code>write</code>	<code>std_logic</code>
<code>writedata</code>	<code>std_logic_vector</code>
<code>read</code>	<code>std_logic</code>
<code>lock</code>	<code>std_logic</code>
<code>readdata</code>	<code>std_logic_vector</code>
<code>response</code>	<code>std_logic_vector</code>
<code>waitrequest</code>	<code>std_logic</code>
<code>readdatavalid</code>	<code>std_logic</code>
<code>irq</code>	<code>std_logic</code>



UVVM™

## 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

## 1 BFM procedure details and examples

Procedure	Description
<b>avalon_mm_write()</b>	<p><b>avalon_mm_write(addr_value, data_value, msg, clk, avalon_mm_if, [byte_enable,] [scope, [msg_id_panel, [config]]])</b></p> <p>The <code>avalon_mm_write()</code> 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.</p> <ul style="list-style-type: none"> <li>- If the <code>byte_enable</code> argument is not used, it will be set to all '1', i.e. all bytes are used.</li> <li>- The <code>avalon_mm_write()</code> 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.</li> <li>- The default value of <code>scope</code> is <code>C_SCOPE</code> ("AVALON MM BFM")</li> <li>- The default value of <code>msg_id_panel</code> is <code>shared_msg_id_panel</code>, defined in <code>UVVM-Util</code>.</li> <li>- The default value of <code>config</code> is <code>C_AVALON_MM_BFM_CONFIG_DEFAULT</code>, see table on the first page.</li> <li>- A log message is written after procedure completes if <code>ID_BFM</code> ID is enabled for the specified message ID panel.</li> </ul> <p>The procedure reports an alert if:</p> <ul style="list-style-type: none"> <li>- waitrequest is enabled for more than 'config.max_wait_cycles' clock cycles (alert level: 'config.max_wait_cycles_severity')</li> </ul> <p>Examples:</p> <pre>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);</pre> <p>Suggested usage (requires local overload, see section 5):</p> <pre>avalon_mm_write(C_ADDR_DMA, x"AAFF0055", "Writing data to DMA");</pre>
<b>avalon_mm_read()</b>	<p><b>avalon_mm_read(addr_value, data_value, msg, clk, avalon_mm_if, [scope, [msg_id_panel, [config, [proc_name]]])</b></p> <p>The <code>avalon_mm_read()</code> 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.</p> <ul style="list-style-type: none"> <li>- The <code>avalon_mm_read()</code> procedure supports pipelining/fixed wait-states, <code>readdatavalid</code> and/or <code>waitrequest</code>, set by the <code>config</code> parameter.             <ul style="list-style-type: none"> <li>- The maximum number of wait cycles while waiting for <code>readdatavalid</code> is given in 'config.max_wait_cycles'</li> <li>- The maximum number of cycles acceptable to be stalled by <code>waitrequest</code> is given in 'config.max_wait_cycles'</li> <li>- If <code>use_waitrequest</code> and <code>use_readdatavalid</code> are disabled in the <code>config</code>, the read procedure will use the <code>num_wait_states</code> as <code>readWaitTime</code>.</li> </ul> </li> <li>- The default value of <code>scope</code> is <code>C_SCOPE</code> ("AVALON MM BFM")</li> <li>- The default value of <code>msg_id_panel</code> is <code>shared_msg_id_panel</code>, defined in <code>UVVM-Util</code>.</li> <li>- The default value of <code>config</code> is <code>C_AVALON_MM_BFM_CONFIG_DEFAULT</code>, see table on the first page.</li> <li>- The default value of <code>proc_name</code> is "avalon_mm_read". This argument is intended to be used internally, when procedure is called by <code>avalon_mm_check()</code>.</li> <li>- A log message is written if <code>ID_BFM</code> ID is enabled for the specified message ID panel. This will only occur if the argument <code>proc_name</code> is left unchanged.</li> <li>- The BFM can be configured to use <code>waitrequest</code> and <code>readdatavalid</code> in the <code>config</code> parameter.</li> </ul> <p>The procedure reports an alert if:</p> <ul style="list-style-type: none"> <li>- <code>waitrequest</code> is enabled for more than 'config.max_wait_cycles' clock cycles (alert level: 'config.max_wait_cycles_severity')</li> <li>- <code>readdatavalid</code> is not set active for more than 'config.max_wait_cycles' clock cycles (alert level: 'config.max_wait_cycles_severity')</li> </ul>

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");
```

## 2 BFM Configuration record

Type name: `t_avalon_mm_bfm_config`

Record element	Type	C_AVALON_MM_BFM_CONFIG_DEFAULT	Description
<code>max_wait_cycles</code>	integer	10	Sets the maximum number of wait cycles before an alert occurs when waiting for readdatavalid or stalling because of waitrequest
<code>max_wait_cycles_severity</code>	<code>t_alert_level</code>	TB_FAILURE	The above timeout will have this severity
<code>clock_period</code>	time	10 ns	Period of the clock signal.
<code>clock_period_margin</code>	time	0 ns	Input clock period margin to specified <code>clock_period</code>
<code>clock_period_severity</code>	<code>t_alert_level</code>	TB_ERROR	The above margin will have this severity
<code>setup_time</code>	time	2.5 ns	Setup time for generated signals. Suggested value is <code>clk_period/4</code> . An alert is reported if <code>setup_time</code> exceed <code>clock_period/2</code> .
<code>hold_time</code>	time	2.5 ns	Hold time for generated signals. Suggested value is <code>clk_period/4</code> . An alert is reported if <code>hold_time</code> exceed <code>clock_period/2</code> .
<code>num_wait_states_read</code>	natural	0	Number of fixed wait states to use for read
<code>num_wait_states_write</code>	natural	0	Number of fixed wait states to use for write
<code>use_waitrequest</code>	boolean	true	Set to true if slave uses waitrequest
<code>use_readdatavalid</code>	boolean	false	Set to true if slave uses readdatavalid
<code>use_response_signal</code>	boolean	true	Whether or not to check the response signal on read
<code>use_begintransfer</code>	boolean	false	Whether or not to use the begintransfer signal.
<code>id_for_bfm</code>	<code>t_msg_id</code>	ID_BFM	The message ID used as a general message ID in the Avalon BFM
<code>id_for_bfm_wait</code>	<code>t_msg_id</code>	ID_BFM_WAIT	The message ID used for logging waits in the Avalon BFM
<code>id_for_bfm_poll</code>	<code>t_msg_id</code>	ID_BFM_POLL	The message ID used for logging polling in the Avalon BFM

## 3 Additional Documentation

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

## 4 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.

### 4.1 Simulator compatibility and setup

This BFM has been compiled and tested with Modelsim version 10.4b and Riviera-PRO version 2015.10.85.

For required simulator setup see UVVM-Util Quick reference.



## 5 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)

### INTELLECTUAL PROPERTY

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.