**AXI4 BFM** –Quick Reference

**BFM**

*axi\_bfm\_pkg.vhd*

|  |  |
| --- | --- |
| axi\_write (awid\_value, awaddr\_value, awlen\_value, awsize\_value, awburst\_value, awlock\_value, awcache\_value, awprot\_value, awqos\_value, awregion\_value, awuser\_value, wdata\_value, wstrb\_value, wuser\_value, buser\_value, bresp\_value, msg, clk, axi\_if, [scope, [msg\_id\_panel, [config]]]) | |
| Example: axi\_write(  awid\_value => x”01”,  awaddr\_value => x"00000004",  awlen\_value => x"01",  awsize\_value => 4,  awburst\_value => INCR,  awlock\_value => NORMAL,  awcache\_value => “0000”,  awprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,  awqos\_value => “0000”,  awregion\_value => “0000”,  awuser\_value => x“01”,  wdata\_value => t\_slv\_array'(x"12345678", x"33333333"),  wstrb\_value => t\_slv\_array'(x"F", x"F"),  wuser\_value => t\_slv\_array'(x"01", x"01"),  buser\_value => v\_buser\_value,  bresp\_value => v\_bresp\_value,  msg => "Writing data to Peripheral 1",  clk => clk,  axi\_if => axi\_if); | ***Suggested usage:*** *axi\_write(*  *awaddr\_value => x"00000004",*  *awlen\_value => x"01",*  *wdata\_value => t\_slv\_array'(x"12345678", x"33333333"),*  *msg => "Writing data to Peripheral 1");*  *-- Suggested usage requires local overload (see section 5)* |



|  |  |
| --- | --- |
| axi\_read (arid\_value, araddr\_value, arlen\_value, arsize\_value, arburst\_value, arlock\_value, arcache\_value, arprot\_value, arqos\_value, arregion\_value, aruser\_value, rdata\_value, rresp\_value, ruser\_value, msg, clk, axi\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]]) | |
| Example: axi\_read(  arid\_value => x“01”,  araddr\_value => x”00000004”,  arlen\_value => x”01”,  arsize\_value => 4,  arburst\_value => INCR,  arlock\_value => NORMAL,  arcache\_value => “0000”,  arprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,  arqos\_value => “0000”,  arregion\_value => “0000”,  aruser\_value => x“01”,  rdata\_value => v\_rdata\_value,  rresp\_value => v\_rresp\_value,  ruser\_value => v\_ruser\_value,  msg => “Read from Peripheral 1”,  clk => clk,  axi\_if => axi\_if); | ***Suggested usage:*** *axi\_read(*  *araddr\_value => C\_ADDR\_IO,*  *arlen\_value => x”01”,*  *rdata\_value => v\_data\_out,*  *msg => “Read from IO”);*  *-- Suggested usage requires local overload (see section 5)* |

|  |  |
| --- | --- |
| axi\_check (arid\_value, araddr\_value, arlen\_value, arsize\_value, arburst\_value, arlock\_value, arcache\_value, arprot\_value, arqos\_value, arregion\_value, aruser\_value, rdata\_exp, rresp\_exp, ruser\_exp, msg, clk, axi\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]]) | |
| Example: axi\_check(  arid\_value => x“01”,  araddr\_value => x”00000004”,  arlen\_value => x”01”,  arsize\_value => 4,  arburst\_value => INCR,  arlock\_value => NORMAL,  arcache\_value => “0000”,  arprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,  arqos\_value => “0000”,  arregion\_value => “0000”,  aruser\_value => x“01”,  rdata\_exp => t\_slv\_array'(x"12345678", x"33333333"),  rresp\_exp => t\_slv\_array’("00", "00"),  ruser\_exp => t\_slv\_array’(x"00", x"00"),  msg => “Check data from Peripheral 1”,  clk => clk,  axi\_if => axi\_if); | ***Suggested usage:*** *axi\_check(*  *araddr\_value => C\_ADDR\_IO,*  *arlen\_value => “01”,*  *rdata\_exp => t\_slv\_array’*(x"12345678", x"33333*333"),*  *msg => “Checking data from Peripheral 1”);*  *-- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| init\_axi\_if\_signals (addr\_width, data\_width, id\_width, user\_width) |
| Example: axi\_if <= init\_axi\_if\_signals(addr\_width, data\_width, id\_width, user\_width); |

BFM Configuration record ´**t\_axi\_bfm\_config´**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **C\_AXI\_BFM\_CONFIG\_DEFAULT** |
| max\_wait\_cycles | natural | 1000 |
| 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\_aw\_pipe\_stages | natural | 1 |
| num\_w\_pipe\_stages | natural | 1 |
| num\_ar\_pipe\_stages | natural | 1 |
| num\_r\_pipe\_stages | natural | 1 |
| num\_b\_pipe\_stages | natural | 1 |
| 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 |

BFM non-signal parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| axid\_value | std\_logic\_vector | x”01” | Identification tag for a read or write transaction |
| axaddr\_value | unsigned | x”125A” | The address of the first transfer in a read or write transaction |
| axlen\_value | unsigned(7 downto 0) | x”01” | The number of data transfers in a read or write transactions |
| axsize\_value | Integer range 1 to 128 | 4 | The number of bytes in each data transfer in a read or write transaction (Must be a power of two) |
| axburst\_value | t\_axburst | INCR | Burst type, indicates how address changes between each transfer in a read or write transaction |
| axlock\_value | t\_axlock | NORMAL | Provides information about the atomic characteristics of a read or write transaction |
| axcache\_value | std\_logic\_vector(3 downto 0) | “0000” | Indicates how a read or write transaction is required to progress through a system |
| axprot\_value | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | Protection attributes of a read or write transaction. Privilege, security level and access type |
| axqos\_value | std\_logic\_vector(3 downto 0) | “0000” | Quality of Service identifier for a read or write transaction |
| axregion\_value | std\_logic\_vector(3 downto 0) | “0000” | Region indicator for a read or write transaction |
| axuser\_value | std\_logic\_vector | x”01” | User-defined extension for the read or write address channel |
| wdata\_value | t\_slv\_array | t\_slv\_array’(x”20D3”, x”1234”) | Array of data values to be written to the addressed registers |
| wstrb\_value | t\_slv\_array | t\_slv\_array’(”1111”, ”1111”) | Array of write strobes, indicates which byte lanes hold valid data. (all ‘1’ means all bytes are updated) |
| wuser\_value | t\_slv\_array | t\_slv\_array’(x”00”, x”01”) | Array of user-defined extension for the write data channel |
| buser\_value | std\_logic\_vector | v\_buser\_value | Output variable containing the user-defined extension for the write response channel |
| bresp\_value | t\_xresp | v\_bresp\_value | Output variable containing the write response which indicates the status of a write transaction |
| rdata\_value | t\_slv\_array | v\_rdata\_value | Output variable containing an array of read data |
| rresp\_value | t\_xresp\_array | v\_rresp\_value | Output variable containing an array of read responses which indicates the status of a read transfer |
| ruser\_value | t\_slv\_array | v\_ruser\_value | Output variable containing an array of user-defined extensions for the read data channel |
| rdata\_exp | t\_slv\_array | t\_slv\_array’(x”ABCD”, x”1234”) | Array of expected read data values. A mismatch results in an alert ‘alert\_level’ |
| rresp\_exp | t\_xresp\_array | t\_xresp\_array’(OKAY, OKAY) | Array of expected read responses which indicates the status of a read transfer |
| ruser\_exp | t\_slv\_array | t\_slv\_array’(x”01”, x”01”) | Array of expected user-defined extensions for the read data channel |
| 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 | "AXI BFM" | A string describing the scope from which the log/alert originates. In a simple single sequencer typically "AXI BFM". In a verification component typically "AXI\_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\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT | Configuration of BFM behaviour and restrictions. See section 2 for details. |

BFM signal parameters

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| clk | std\_logic | The clock signal used to read and write data in/out of the AXI4 BFM. |
| axi\_if | t\_axi\_if | See table “Signal record ‘axi\_if’” |

Note: All signals are active high. See AXI4 documentation for protocol description.

For more information on the AXI4 signals, please see the AXI4 specification.

Signal record ‘axi\_if’

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Record element** | **Type** | | write\_address\_channel | t\_axi\_write\_address\_channel | | write\_data\_channel | t\_axi\_write\_data\_channel | | write\_response\_channel | t\_axi\_write\_response\_channel | | read\_address\_channel | t\_axi\_read\_address\_channel | | read\_data\_channel | t\_axi\_read\_data\_channel | |  |
| |  |  | | --- | --- | | **Record element** | **Type** | | **write\_address\_channel** | **t\_axi\_write\_address\_channel** | | awid | std\_logic\_vector | | awaddr | std\_logic\_vector | | awlen | std\_logic\_vector(7 downto 0) | | awsize | std\_logic\_vector(2 downto 0) | | awburst | std\_logic\_vector(1 downto 0) | | awlock | std\_logic | | awcache | std\_logic\_vector(3 downto 0) | | awprot | std\_logic\_vector(2 downto 0) | | awqos | std\_logic\_vector(3 downto 0) | | awregion | std\_logic\_vector(3 downto 0) | | awuser | std\_logic\_vector | | awvalid | std\_logic | | awready | std\_logic | | |  |  | | --- | --- | | **Record element** | **Type** | | **write\_data\_channel** | **t\_axi\_write\_data\_channel** | | wdata | std\_logic\_vector | | wstrn | std\_logic\_vector | | wlast | std\_logic | | wuser | std\_logic\_vector | | wvalid | std\_logic | | wready | std\_logic | |
| |  |  | | --- | --- | | **Record element** | **Type** | | **write\_response\_channel** | **t\_axi\_write\_response\_channel** | | bid | std\_logic\_vector | | bresp | std\_logic\_vector(1 downto 0) | | buser | std\_logic\_vector | | bvalid | std\_logic | | bready | std\_logic | |
| |  |  | | --- | --- | | **Record element** | **Type** | | **read\_address\_channel** | **t\_axi\_read\_address\_channel** | | arid | std\_logic\_vector | | araddr | std\_logic\_vector | | arlen | std\_logic\_vector(7 downto 0) | | arsize | std\_logic\_vector(2 downto 0) | | arburst | std\_logic\_vector(1 downto 0) | | arlock | std\_logic | | arcache | std\_logic\_vector(3 downto 0) | | arprot | std\_logic\_vector(2 downto 0) | | arqos | std\_logic\_vector(3 downto 0) | | arregion | std\_logic\_vector(3 downto 0) | | aruser | std\_logic\_vector | | arvalid | std\_logic | | arready | std\_logic | | |  |  | | --- | --- | | **Record element** | **Type** | | **read\_data\_channel** | **t\_axi\_read\_data\_channel** | | rid | std\_logic\_vector | | rdata | std\_logic\_vector | | rresp | std\_logic\_vector(1 downto 0) | | rlast | std\_logic | | ruser | std\_logic\_vector | | rvalid | std\_logic | | rready | std\_logic | |

AXI parameter record types

|  |  |
| --- | --- |
| **Type name** | **Allowed value** |
| t\_axburst | FIXED |
|  | INCR |
|  | WRAP |
| t\_axlock | NORMAL |
|  | EXCLUSIVE |
| t\_axprot | UNPRIVILEGED\_NONSECURE\_DATA |
|  | UNPRIVILEGED\_NONSECURE\_INSTRUCTION |
|  | UNPRIVILEGED\_SECURE\_DATA |
|  | UNPRIVILEGED\_SECURE\_INSTRUCTION |
|  | PRIVILEGED\_NONSECURE\_DATA |
|  | PRIVILEGED\_NONSECURE\_INSTRUCTION |
|  | PRIVILEGED\_SECURE\_DATA |
|  | PRIVILEGED\_SECURE\_INSTRUCTION |
| t\_xresp | OKAY |
|  | EXOKAY |
|  | SLVERR |
|  | DECERR |

BFM details

# BFM procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **axi\_write()** | **axi\_write(awid\_value, awaddr\_value, awlen\_value, awsize\_value, awburst\_value, awlock\_value, awcache\_value, awprot\_value, awqos\_value, awregion\_value, awuser\_value, wdata\_value, wstrb\_value, wuser\_value, buser\_value, bresp\_value, msg, clk, axi\_if, [scope, [msg\_id\_panel, [config]]])**  The axi\_write() procedure writes the given data to the given address of the DUT, using the AXI4 protocol. For protocol details, see the AXI4 specification.   * A log message is written if ID\_BFM ID is enabled for the specified message ID panel. * The default values of the various parameters are shown in the table below.  |  |  |  | | --- | --- | --- | | **Parameter name** | **Type** | **Default value** | | awid\_value | std\_logic\_vector | 0 | | awaddr\_value | unsigned | None | | awlen\_value | unsigned(7 downto 0) | 0 | | awsize\_value | Integer range 1 to 128 | 4 | | awburst\_value | t\_axburst | INCR | | awlock\_value | t\_axlock | NORMAL | | awcache\_value | std\_logic\_vector(3 downto 0) | 0 | | awprot\_value | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | | awqos\_value | std\_logic\_vector(3 downto 0) | 0 | | awregion\_value | std\_logic\_vector(3 downto 0) | 0 | | awuser\_value | std\_logic\_vector | 0 | | wdata\_value | t\_slv\_array | None | | wstrb\_value | t\_slv\_array | (others => ‘1’) for all words | | wuser\_value | t\_slv\_array | 0 for all words | | buser\_value | std\_logic\_vector | None | | bresp\_value | std\_logic\_vector(1 downto 0) | None | | msg | string | None | | clk | std\_logic | None | | axi\_if | t\_axi\_if | None | | scope | string | C\_SCOPE (“AXI BFM”) | | msg\_id\_panel | t\_msg\_id\_panel | shared\_msg\_id\_panel | | config | t\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT |   The procedure reports an alert if:   * wready does not occur within max\_wait\_cycles clock cycles (alert level: max\_wait\_cycles\_severity, set in the config) * awready does not occur within max\_wait\_cycles clock cycles (alert level: max\_wait\_cycles\_severity, set in the config) * bvalid is not set within max\_wait\_cycles clock cycles (alert level: max\_wait\_cycles\_severity, set in the config)     Examples:  axi\_write(  awid\_value => x”01”,  awaddr\_value => x"00000004",  awlen\_value => x"01",  awsize\_value => 4,  awburst\_value => INCR,  awlock\_value => NORMAL,  awcache\_value => “0000”,  awprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,  awqos\_value => “0000”,  awregion\_value => “0000”,  awuser\_value => x“01”,  wdata\_value => t\_slv\_array'(x"12345678", x"33333333"),  wstrb\_value => t\_slv\_array'(x"F", x"F"),  wuser\_value => t\_slv\_array'(x"01", x"01"),  buser\_value => v\_buser\_value,  bresp\_value => v\_bresp\_value,  msg => "Writing data to Peripheral 1",  clk => clk,  axi\_if => axi\_if,  scope => C\_SCOPE,  msg\_id\_panel => shared\_msg\_id\_panel,  config => C\_AXI\_BFM\_CONFIG\_DEFAULT);  axi\_write(  awaddr\_value => x"00000004",  wdata\_value => t\_slv\_array'(x"12345678", x"33333333"),  buser\_value => v\_buser\_value,  bresp\_value => v\_bresp\_value,  msg => "Writing data to Peripheral 1");  Suggested usage (requires local overload, see section 5):  axi\_write(C\_ADDR\_DMA, x”AAAA”, “Writing data to DMA”);  axi\_write(C\_ADDR\_MEMORY, x”FF”, v\_data\_array, “Writing 256 data words to MEMORY”); |
| **axi\_read()** | **axi\_read(arid\_value, araddr\_value, arlen\_value, arsize\_value, arburst\_value, arlock\_value, arcache\_value, arprot\_value, arqos\_value, arregion\_value, aruser\_value, rdata\_value, rresp\_value, ruser\_value, msg, clk, axi\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]])**  The axi\_read() procedure reads data from the DUT at the given address, using the AXI4 protocol. For protocol details, see the AXI4 specification. The read data is placed on the output ‘rdata\_value’ when the read has completed.   * The argument “ext\_proc\_call” is intended to be used internally, when the procedure is called by axi\_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 default values of the various parameters are shown in the table below.  |  |  |  | | --- | --- | --- | | **Parameter name** | **Type** | **Default value** | | arid\_value | std\_logic\_vector | 0 | | araddr\_value | unsigned | None | | arlen\_value | unsigned(7 downto 0) | 0 | | arsize\_value | Integer range 1 to 128 | 4 | | arburst\_value | t\_axburst | INCR | | arlock\_value | t\_axlock | NORMAL | | arcache\_value | std\_logic\_vector(3 downto 0) | 0 | | arprot\_value | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | | arqos\_value | std\_logic\_vector(3 downto 0) | 0 | | arregion\_value | std\_logic\_vector(3 downto 0) | 0 | | aruser\_value | std\_logic\_vector | 0 | | rdata\_value | t\_slv\_array | None | | rresp\_value | t\_xresp\_array | None | | ruser\_value | t\_slv\_array | None | | msg | string | None | | clk | std\_logic | None | | axi\_if | t\_axi\_if | None | | scope | string | C\_SCOPE (“AXI BFM”) | | msg\_id\_panel | t\_msg\_id\_panel | shared\_msg\_id\_panel | | config | t\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT | | ext\_proc\_call | string | “” |   The procedure reports an alert if:   * The received rid is different from the transmitted arid\_value * arready does not occur within max\_wait\_cycles clock cycles (alert level: max\_wait\_cycles\_severity, set in the config) * rvalid is not set within max\_wait\_cycles clock cycles (alert level: max\_wait\_cycles\_severity, set in the config)   Examples:  axi\_read(  arid\_value => x“01”,  araddr\_value => x”00000004”,  arlen\_value => x”01”,  arsize\_value => 4,  arburst\_value => INCR,  arlock\_value => NORMAL,  arcache\_value => “0000”,  arprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,  arqos\_value => “0000”,  arregion\_value => “0000”,  aruser\_value => x“01”,  rdata\_value => v\_rdata\_value,  rresp\_value => v\_rresp\_value,  ruser\_value => v\_ruser\_value,  msg => “Read from Peripheral 1”,  clk => clk,  axi\_if => axi\_if,  scope => C\_SCOPE,  msg\_id\_panel => shared\_msg\_id\_panel,  config => C\_AXI\_BFM\_CONFIG\_DEFAULT);  axi\_read(  araddr\_value => x”00000004”,  rdata\_value => v\_rdata\_value,  rresp\_value => v\_rresp\_value,  ruser\_value => v\_ruser\_value,  msg => “Read from Peripheral 1”,  clk => clk,  axi\_if => axi\_if);  Suggested usage (requires local overload, see section 5):  axi\_read(C\_ADDR\_IO, v\_data\_out, “Reading from IO device”);  axi\_read(C\_ADDR\_MEMORY, x”FF”, v\_data\_array\_out, “Reading 256 data words from MEMORY”); |
| **axi\_check()** | **axi\_check(arid\_value, araddr\_value, arlen\_value, arsize\_value, arburst\_value, arlock\_value, arcache\_value, arprot\_value, arqos\_value, arregion\_value, aruser\_value, rdata\_exp, rresp\_exp, ruser\_exp, msg, clk, axi\_if, [alert\_level, [scope, [msg\_id\_panel, [config]]]])**  The axi\_check() procedure reads data from the DUT at the given address, using the AXI4 protocol. For protocol details, see the AXI4 specification. After reading data from the AXI4 bus, the read data is compared with the expected data, ‘rdata\_exp’.   * If the check was successful, and the read data matches the expected data, a log message is written with ID\_BFM ID (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.  |  |  |  | | --- | --- | --- | | **Parameter name** | **Type** | **Default value** | | arid\_value | std\_logic\_vector | 0 | | araddr\_value | unsigned | None | | arlen\_value | unsigned(7 downto 0) | 0 | | arsize\_value | Integer range 1 to 128 | 4 | | arburst\_value | t\_axburst | INCR | | arlock\_value | t\_axlock | NORMAL | | arcache\_value | std\_logic\_vector(3 downto 0) | 0 | | arprot\_value | t\_axprot | UNPRIVILEGED\_UNSECURE\_DATA | | arqos\_value | std\_logic\_vector(3 downto 0) | 0 | | arregion\_value | std\_logic\_vector(3 downto 0) | 0 | | aruser\_value | std\_logic\_vector | 0 | | rdata\_exp | t\_slv\_array | None | | rresp\_exp | t\_xresp\_array | OKAY for all words | | ruser\_exp | t\_slv\_array | 0 for all words | | msg | string | None | | clk | std\_logic | None | | axi\_if | t\_axi\_if | None | | alert\_level | t\_alert\_level | ERROR | | scope | string | C\_SCOPE (“AXI BFM”) | | msg\_id\_panel | t\_msg\_id\_panel | shared\_msg\_id\_panel | | config | t\_axi\_bfm\_config | C\_AXI\_BFM\_CONFIG\_DEFAULT |   The procedure also report alerts for the same conditions as the axi\_read() procedure.  Examples:  axi\_check(  arid\_value => x“01”,  araddr\_value => x”00000004”,  arlen\_value => x”01”,  arsize\_value => 4,  arburst\_value => INCR,  arlock\_value => NORMAL,  arcache\_value => “0000”,  arprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,  arqos\_value => “0000”,  arregion\_value => “0000”,  aruser\_value => x“01”,  rdata\_exp => t\_slv\_array'(x"12345678", x"33333333"),  rresp\_exp => t\_xresp\_array’(OKAY, OKAY),  ruser\_exp => t\_slv\_array’(x"00", x"00"),  msg => “Check data from Peripheral 1”,  clk => clk,  axi\_if => axi\_if,  alert\_level => ERROR,  scope => C\_SCOPE,  msg\_id\_panel => shared\_msg\_id\_panel,  config => C\_AXI\_BFM\_CONFIG\_DEFAULT);  axi\_check(  araddr\_value => x”00000004”,  rdata\_exp => v\_rdata\_exp,  msg => “Check data from Peripheral 1”,  clk => clk,  axi\_if => axi\_if);  Suggested usage (requires local overload, see section 5):  axi\_check(C\_ADDR\_UART\_RX, x”3B”, “Checking data in UART RX register”);  axi\_check(C\_ADDR\_MEMORY, x”FF”, v\_rdata\_exp\_array, “Checking 256 data words from MEMORY”); |
| **init\_axi\_if\_signals()** | **init\_axi\_if\_signals(addr\_width, data\_width, id\_width, user\_width)**  This function initializes the AXI4 interface. All the BFM outputs are set to zeros ('0') and BFM inputs are set to 'Z'.  Note: This function assumes that awid, bid, arid and rid shares a common width (id\_width) and that awuser, buser, aruser, ruser also share a common width (user\_width)  Example:  axi\_if <= init\_axi\_if\_signals(addr\_width, data\_width, id\_width, user\_width); |

# BFM Configuration record

Type name: t\_axi\_bfm\_config

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_AXI\_BFM\_CONFIG\_DEFAULT** | **Description** |
| max\_wait\_cycles | natural | 10 | Used for setting the maximum cycles to wait before an alert is issued when waiting for ready and valid signals from the DUT. |
| 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\_margin\_severity | t\_alert\_level | TB\_ERROR | The above margin will have the 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. |
| num\_aw\_pipe\_stages | natural | 1 | Write Address Channel pipeline steps |
| num\_w\_pipe\_stages | natural | 1 | Write Data Channel pipeline steps |
| num\_ar\_pipe\_stages | natural | 1 | Read Address Channel pipeline steps |
| num\_r\_pipe\_stages | natural | 1 | Read Data Channel pipeline steps |
| num\_b\_pipe\_stages | natural | 1 | Response Channel pipeline steps |
| id\_for\_bfm | t\_msg\_id | ID\_BFM | The message ID used as a general message ID in the AXI BFM |
| id\_for\_bfm\_wait | t\_msg\_id | ID\_BFM\_WAIT | The message ID used for logging waits in the AXI BFM |
| id\_for\_bfm\_poll | t\_msg\_id | ID\_BFM\_POLL | The message ID used for logging polling in the AXI BFM |

# Additional Documentation

For additional documentation on the AXI4 standard, please see the AXI4 specification “AMBA® AXI™ and ACE™ Protocol Specification”, available from ARM.

# Compilation

The AXI4 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 axi\_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.

axi\_write(C\_ADDR\_PERIPHERAL\_1, C\_TEST\_DATA, “Sending data to Peripheral 1”);

rather than

axi\_write(

awid\_value => x”01”,

awaddr\_value => x"00000004",

awlen\_value => x"01",

awsize\_value => 4,

awburst\_value => INCR,

awlock\_value => NORMAL,

awcache\_value => “0000”,

awprot\_value => UNPRIVILEGED\_UNSECURE\_DATA,

awqos\_value => “0000”,

awregion\_value => “0000”,

awuser\_value => x“01”,

wdata\_value => t\_slv\_array'(x"12345678", x"33333333"),

wstrb\_value => t\_slv\_array'(x"F", x"F"),

wuser\_value => t\_slv\_array'(x"01", x"01"),

buser\_value => v\_buser\_value,

bresp\_value => v\_bresp\_value,

msg => "Writing data to Peripheral 1",

clk => clk,

axi\_if => axi\_if,

scope => C\_SCOPE,

msg\_id\_panel => shared\_msg\_id\_panel,

config => C\_AXI\_BFM\_CONFIG\_DEFAULT);

By defining the local overload as e.g.:

procedure axi\_write(

constant addr\_value : in unsigned;

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

constant msg : in string

) is

variable v\_buser\_value : std\_logic\_vector(C\_USER\_WIDTH-1 downto 0);

variable v\_bresp\_value : t\_xresp;

begin

axi\_write(

awid\_value => x"00", -- Setting a default value

awaddr\_value => addr\_value, -- keep as is

awlen\_value => x"00", -- Set to length=1

awsize\_value => 4, -- Setting a default value

awburst\_value => INCR, -- Setting a default value

awlock\_value => NORMAL, -- Setting a default value

awcache\_value => "0000", -- Setting a default value

awprot\_value => UNPRIVILEGED\_UNSECURE\_DATA, -- Setting a default value

awqos\_value => "0000", -- Setting a default value

awregion\_value => "0000", -- Setting a default value

awuser\_value => x"01", -- Setting a default value

wdata\_value => data\_value, -- keep as is

wstrb\_value => x"f" -- Setting a default value

wuser\_value => x"01", -- Setting a default value

buser\_value => v\_buser\_value, -- Assigning to a local variable

bresp\_value => v\_bresp\_value, -- Assigning to a local variable

msg => msg, -- keep as is

clk => clk, -- Clock signal

axi\_if => axi\_if, -- Signal must be visible in local process scope

scope => C\_SCOPE, -- Setting a default value

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

config => C\_AXI\_BFM\_CONFIG\_LOCAL); -- Use locally defined configuration or C\_AXI\_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 msg\_id\_panel to allow dedicated verbosity control

IMPORTANT

This is a simplified Bus Functional Model (BFM) for AXI4.

The given BFM complies with the AXI4 protocol and thus allows a normal access towards an AXI4 interface. This BFM is not AXI4 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**